Vital Ruby

Ruby Training

Intro

Who We Are

EdgeCase
software artisans

Joe O'Brien is a father, speaker, author and developer. Before helping found EdgeCase, LLC, Joe was a developer with ThoughtWorks and spent much of his time working with large J2EE and .NET systems for Fortune 500 companies.

Jim Weirich has been active in the software development world for over twenty-five years, with experience that ranges from real-time data acquisition for jet engine testing to image processing and web services for the financial industry. He is currently the chief scientist for EdgeCase, LLC.

Paul is an Agile/Extreme Programming coach and coder who has worked with many teams to improve their process and 'Agility'. He has been practising Test-Driven Development since 2000, and other Agile practices since 2002. He is an active member of the local Ruby and Agile communities, and co-organiser of the Scotland on Rails conference.

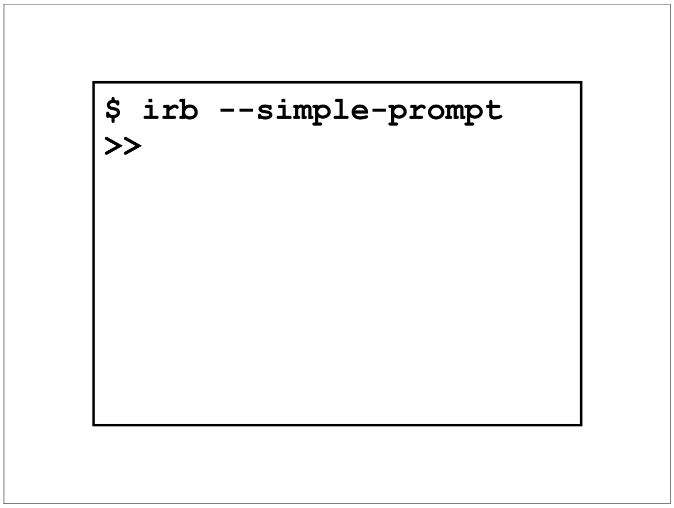
(tentative)

Schedule

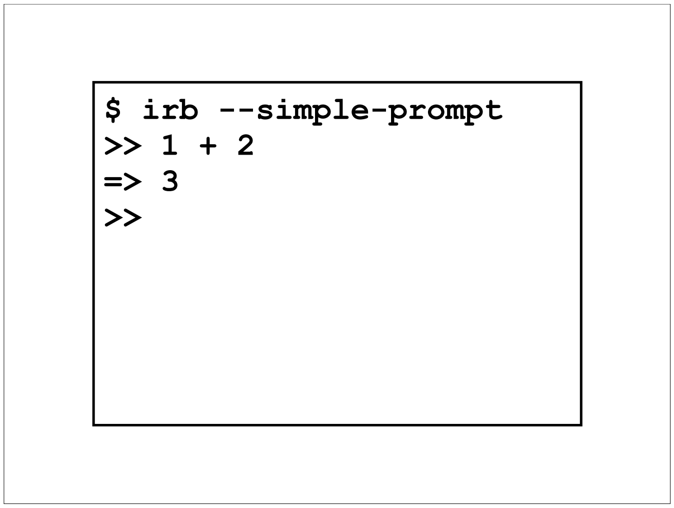
- 9:00 -- Morning Session
- 12:00 -- Lunch
- 1:00 -- Afternoon Session
- 5:00pm -- End of Day

The Basics

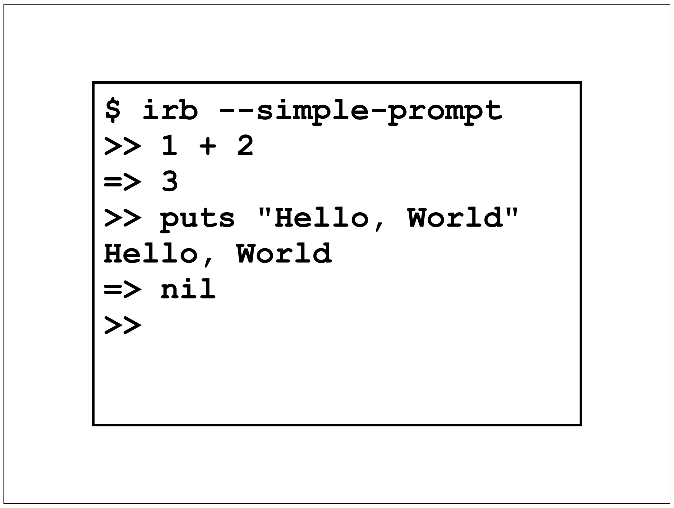
IRB



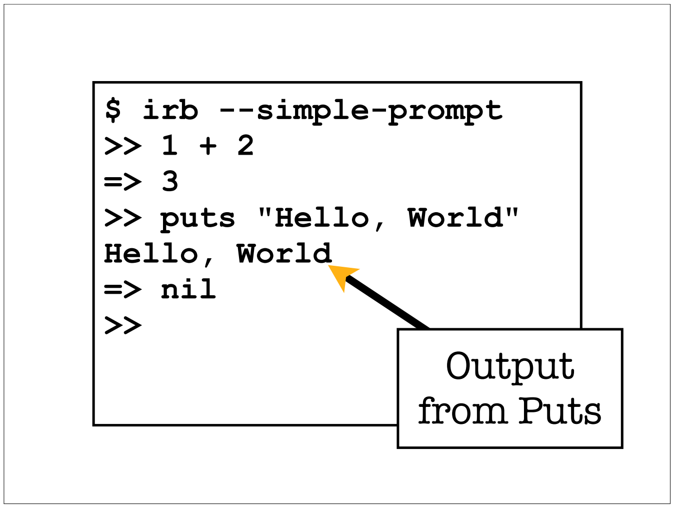
```
$ irb --simple-prompt
>>
```



```
$ irb --simple-prompt
>> 1 + 2
=> 3
>>
```

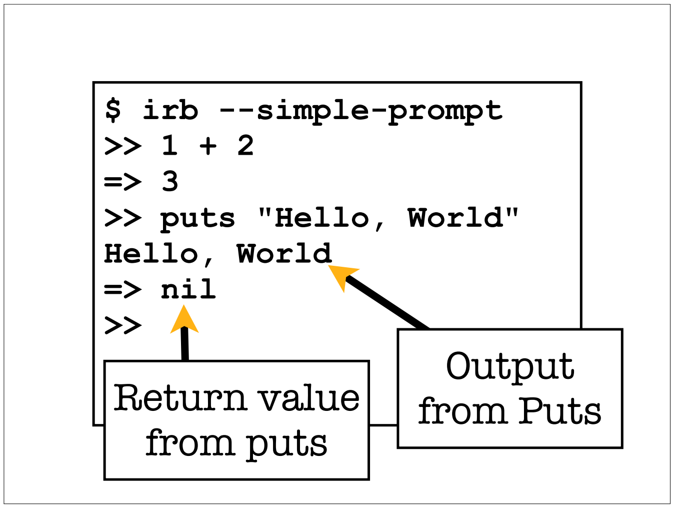


```
$ irb --simple-prompt
>> 1 + 2
=> 3
>> puts "Hello, World"
Hello, World
=> nil
>>
```



```
$ irb --simple-prompt
>> 1 + 2
=> 3
>> puts "Hello, World"
Hello, World
=> nil
>>
```

Output from Puts

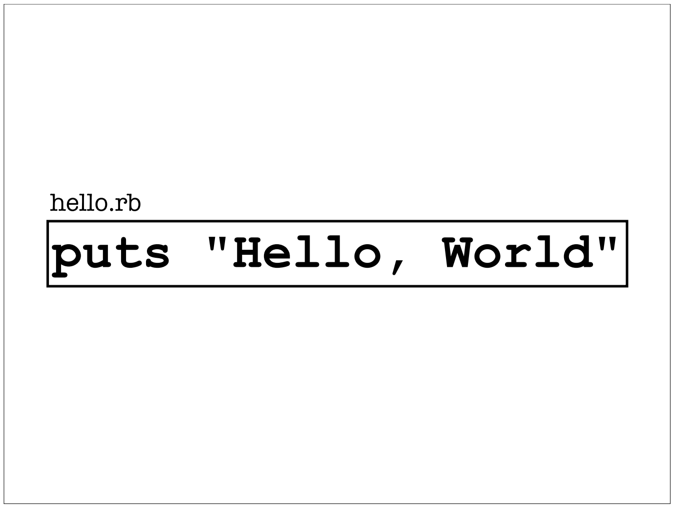


```
$ irb --simple-prompt
>> 1 + 2
=> 3
>> puts "Hello, World"
Hello, World
=> nil
>>
```

Return value from puts

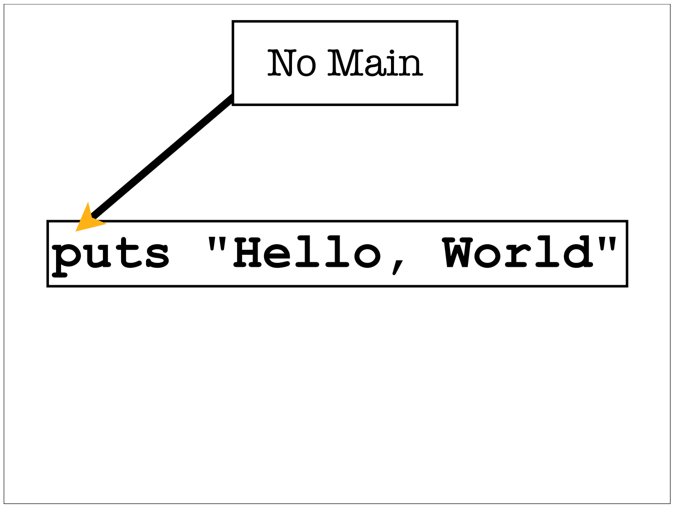
Output from Puts

Files



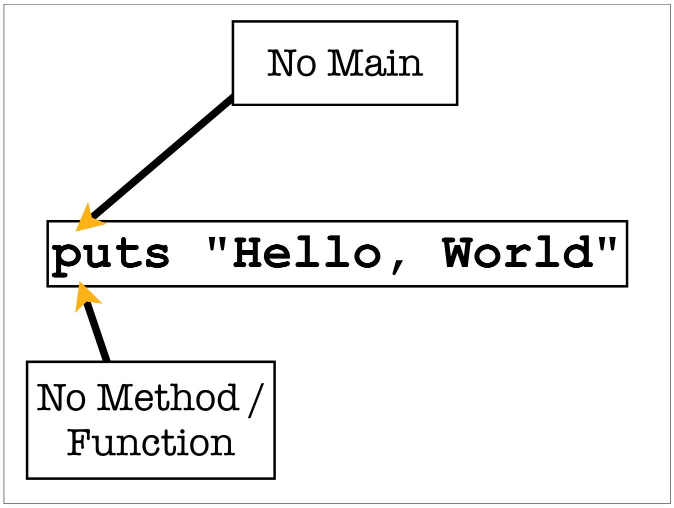
hello.rb

```
puts "Hello, World"
```



No Main

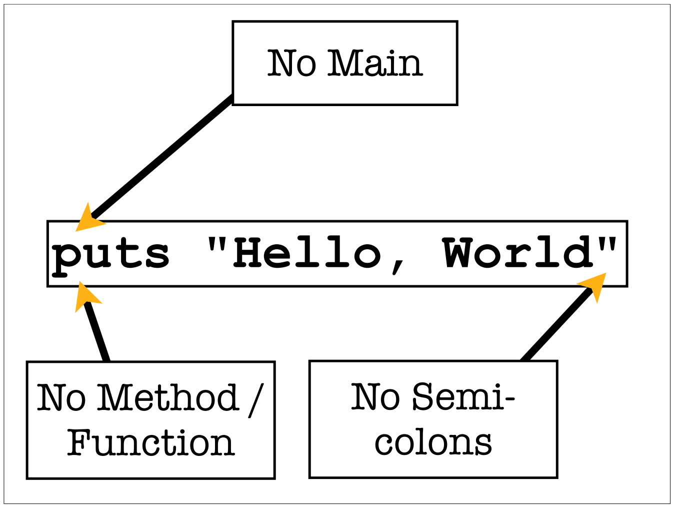
```
puts "Hello, World"
```



No Main

```
puts "Hello, World"
```

No Method / Function



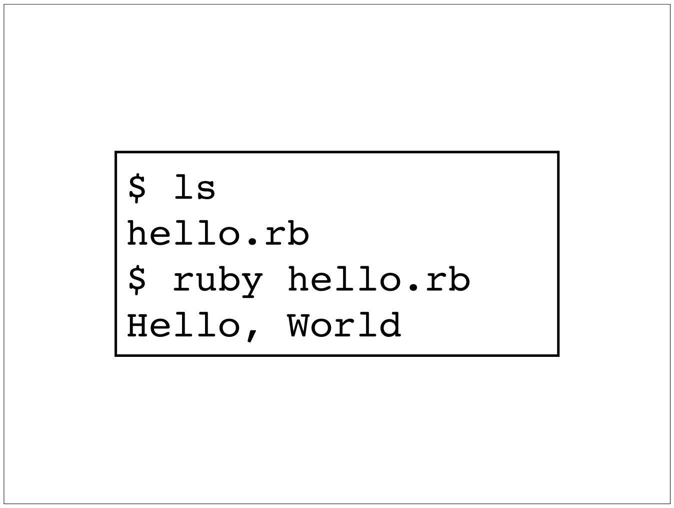
No Main

```
puts "Hello, World"
```

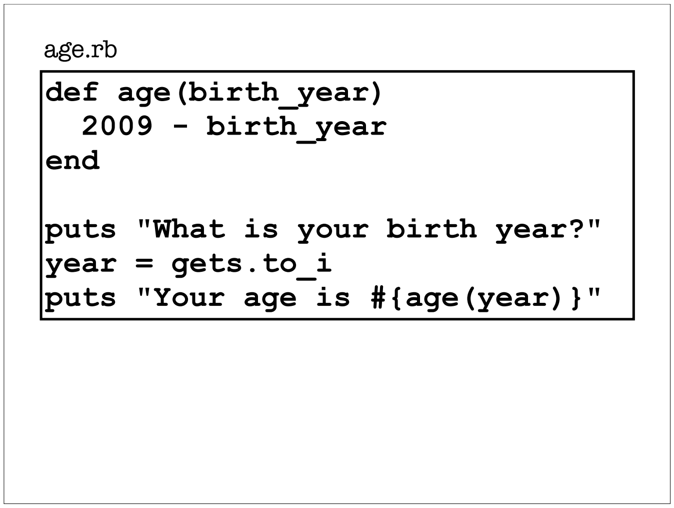
No Method / Function

No Semi-colons

Running



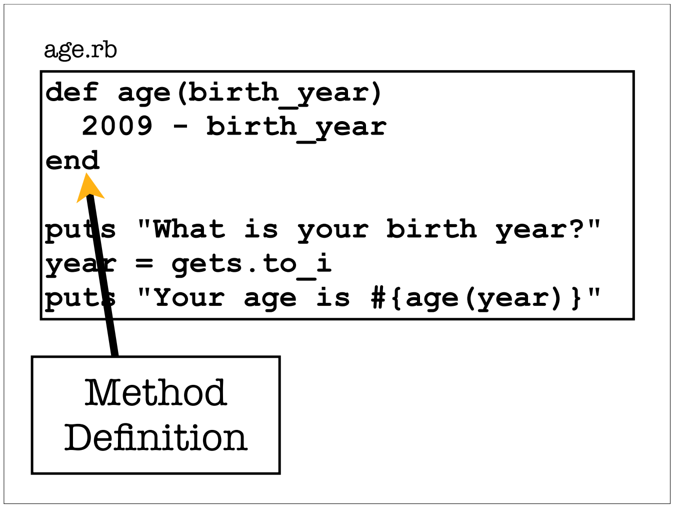
```
$ ls
hello.rb
$ ruby hello.rb
Hello, World
```



age.rb

```
def age(birth_year)
  2009 - birth_year
end

puts "What is your birth year?"
year = gets.to_i
puts "Your age is #{age(year)}"
```

19

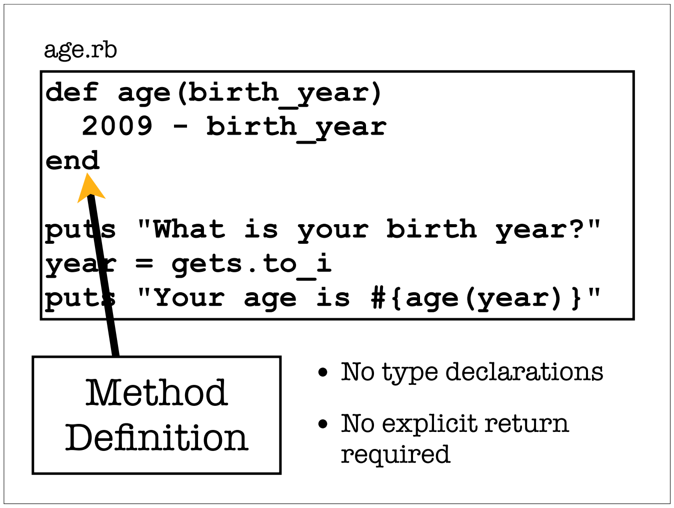


age.rb

```
def age(birth_year)
  2009 - birth_year
end

puts "What is your birth year?"
year = gets.to_i
puts "Your age is #{age(year)}"
```

Method Definition

20



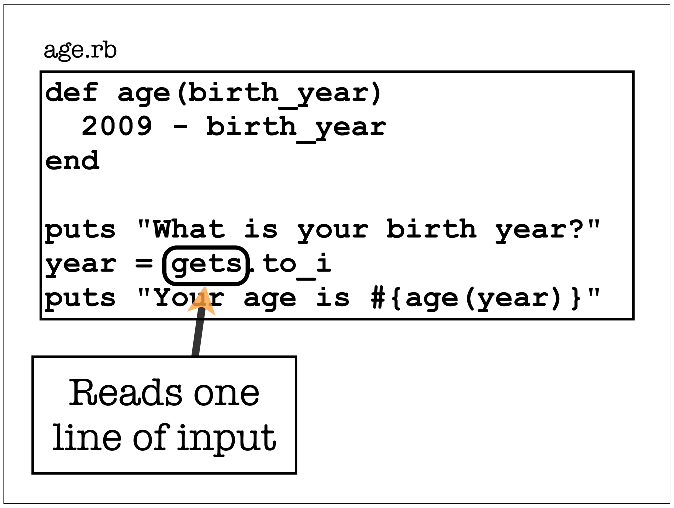
age.rb

```
def age(birth_year)
  2009 - birth_year
end

puts "What is your birth year?"
year = gets.to_i
puts "Your age is #{age(year)}"
```

Method Definition

- No type declarations
- No explicit return required

21

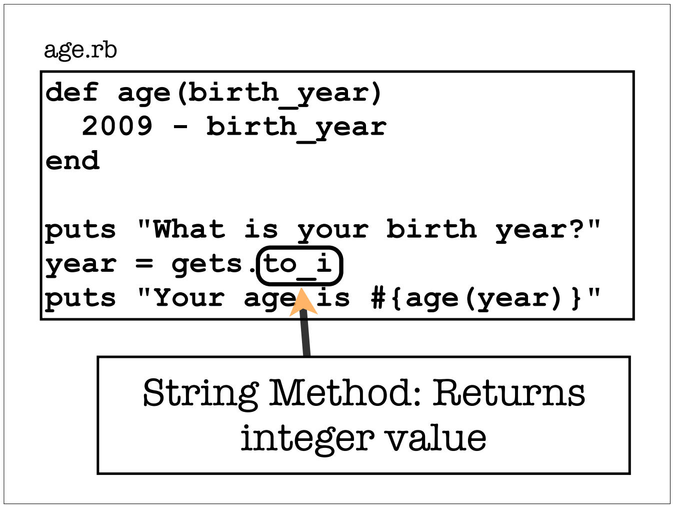


age.rb

```
def age(birth_year)
  2009 - birth_year
end

puts "What is your birth year?"
year = gets.to_i
puts "Your age is #{age(year)}"
```

Reads one line of input

22

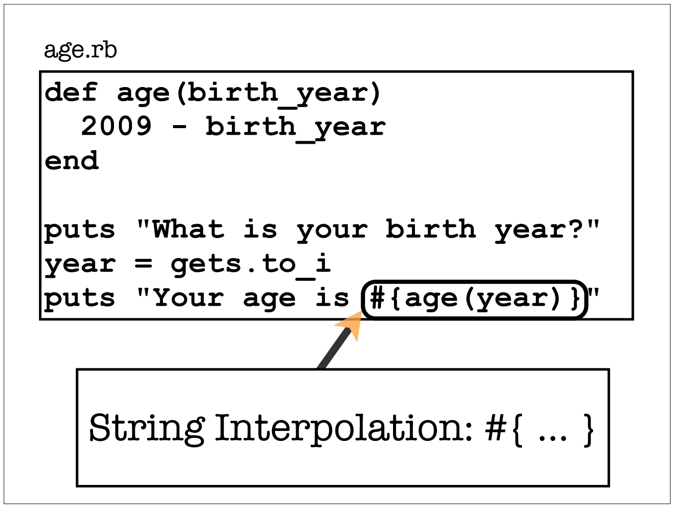


age.rb

```
def age(birth_year)
  2009 - birth_year
end

puts "What is your birth year?"
year = gets.to_i
puts "Your age is #{age(year)}"
```

String Method: Returns integer value

23



age.rb

```
def age(birth_year)
  2009 - birth_year
end

puts "What is your birth year?"
year = gets.to_i
puts "Your age is #{age(year)}"
```

String Interpolation: #{ … }

24

Numerics

```
0, 1, 2, -14      # Fixnum
100_000_000       # Bignum
3.1416            # Float
6.022e23          # Float

10 + (3 * 2)
3.1416.round      # => 3
3.1416.to_s       # "3.1416"
```

```
3 / 2     # => 1
3.0 / 2   # => 1.5
3 / 2.0   # => 1.5
3.0 / 2.0 # => 1.5
```

Integer or Float?

```
a / b
```

Gotchas

```
a.to_f / b      # => Float
a / b.to_f      # => Float
(a/b).to_f      # NO

a.div(b)        # => Integer
```

Use .to_f to get Float
Use .div to get Integer

- Numeric
  - Float
  - Integer
    - Fixnum (< 2**31)
    - Bignum (> 2**31)

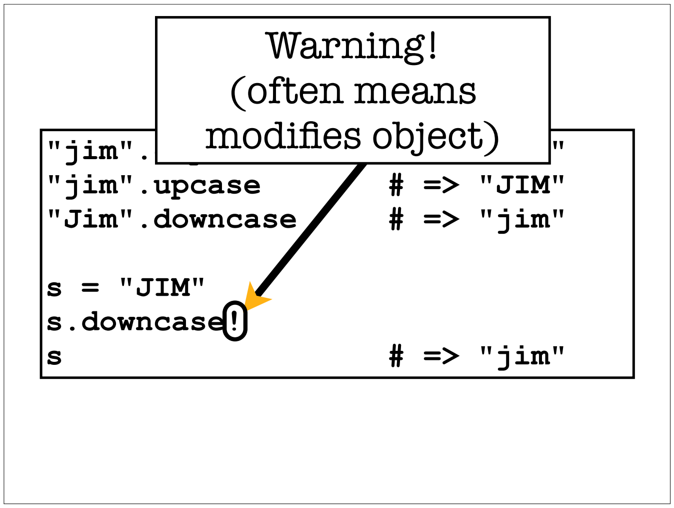
Strings

```
s = "Hello"

s.size    # => 5
s[0]      # => 72 (in Ruby 1.8)
          # => "H" (in Ruby 1.9)
s[1,2]    # => "el" (substring)
s[1..3]   # => "ell"
s[2..-1]  # => "ello"
```

```
str = "2.71828"

str.to_i    # => 2
str.to_f    # => 2.71828

"JIM".to_i  # => 0
```

```
Integer("2")   # => 2
Integer("2.1") # FAIL!
Integer("JIM") # FAIL!
```

```
"jim".capitalize   # => "Jim"
"jim".upcase       # => "JIM"
"Jim".downcase     # => "jim"

s = "JIM"
s.downcase!
s                  # => "jim"
```



```
"jim".                             "
"jim".upcase       # => "JIM"
"Jim".downcase     # => "jim"

s = "JIM"
s.downcase!
s                  # => "jim"
```

Warning!
(often means
modifies object)

```
p = "peanut"
b = "butter"
pb = p + b


p   # => "peanut"
b   # => "butter"
pb  # => "peanutbutter"
```

```
p = "peanut"
b = "butter"
s = p
s += b

p   # => "peanut"
b   # => "butter"
s   # => "peanutbutter"
```

```
p = "peanut"
b = "butter"
s = p
s += b

p   # => "peanut"
b   # => "butter"
s   # => "peanutbutter"
```

a += b
is equivalent to
a = a + b

(also -=, *=, etc)

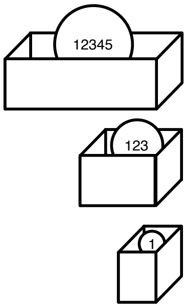
```
p = "peanut"
b = "butter"
s = p
s << b

p   # => "peanutbutter"
b   # => "butter"
s   # => "peanutbutter"
```

(an interlude)

Shoe Boxes
VS
Labels

Shoe Boxes

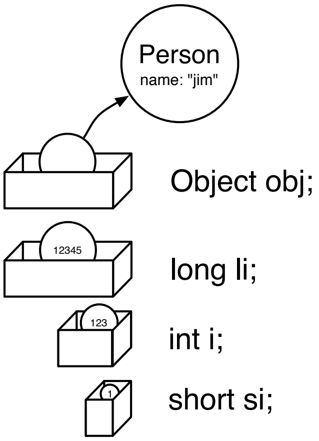
(Java)



long li;

int i;

short si;

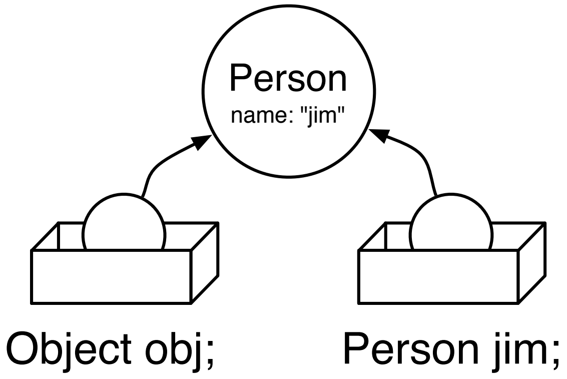


Object obj;

long li;

int i;

short si;

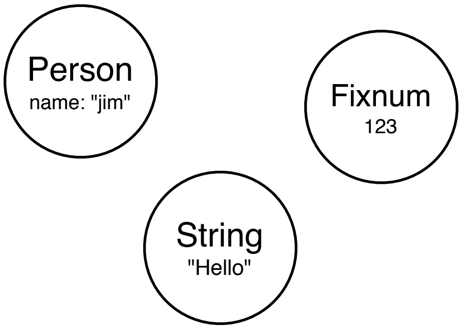


Object obj;

Person jim;

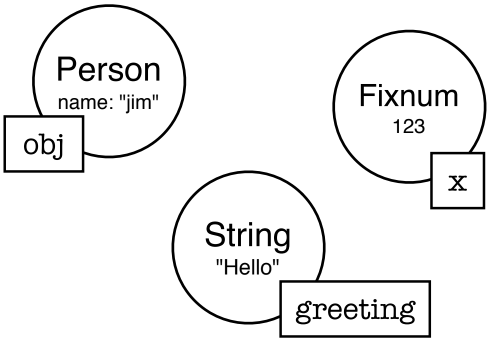
Binding

(Ruby)



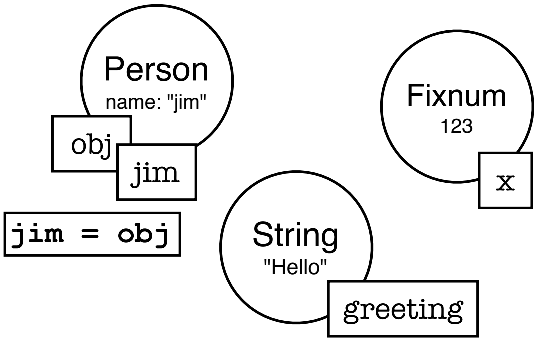
Binding

(Ruby)



Binding

(Ruby)

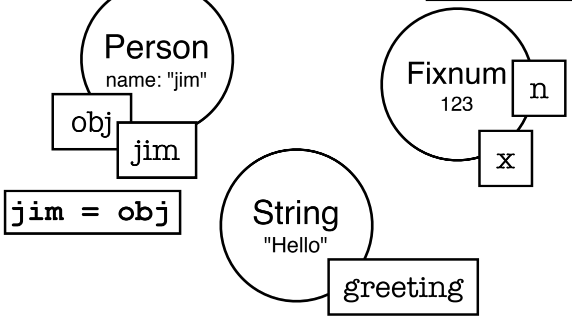


```
jim = obj
```

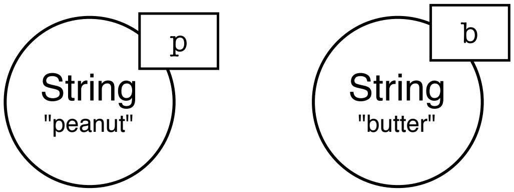
Binding

(Ruby)

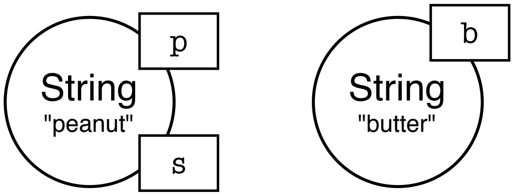
```
def f(n)
...
f(x)
```



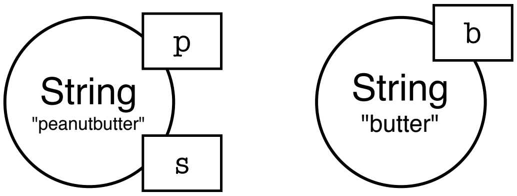
```
jim = obj
```



```
p = "peanut"
b = "butter"
```



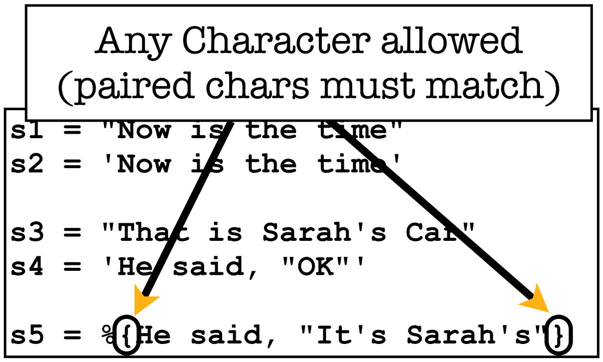
```
p = "peanut"
b = "butter"
s = p
```



```
p = "peanut"
b = "butter"
s = p
s << b
```

(now back to strings)

```
s1 = "Now is the time"
s2 = 'Now is the time'

s3 = "That is Sarah's Car"
s4 = 'He said, "OK"'

s5 = %{He said, "It's Sarah's"}
```



Any Character allowed
(paired chars must match)

```
s1 = "Now is the time"
s2 = 'Now is the time'

s3 = "That is Sarah's Car"
s4 = 'He said, "OK"'

s5 = %{He said, "It's Sarah's"}
```

Try this in IRB ...

```
now = Time.now

"Now is the time: #{now}"
'Now is the time: #{now}'

"\n".size
'\n'.size
```

Interpolation

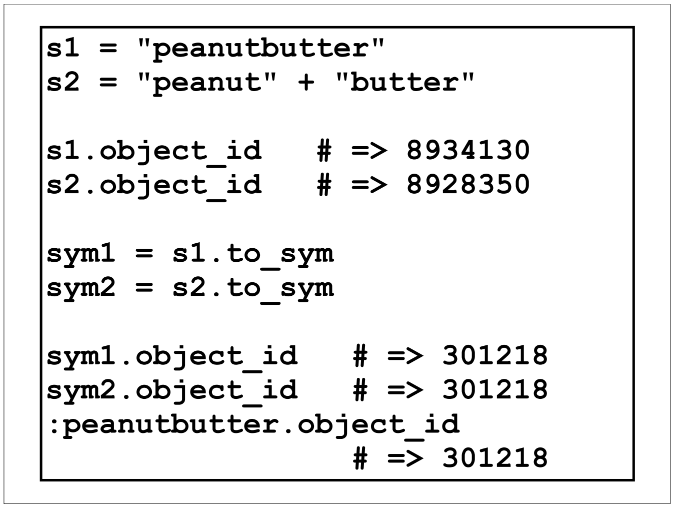
- Interpolating Strings:
  - "str", %[str], %Q[str]
- Non-interpolating Strings:
  - 'str', %q[str]

Symbols

```
sym = :a_symbol

sym.to_s          # => "a_symbol"

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

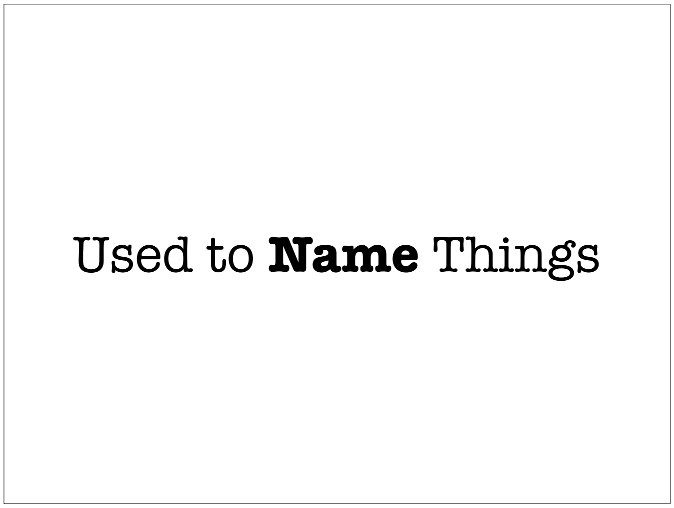
```
s1 = "peanutbutter"
s2 = "peanut" + "butter"

s1.object_id   # => 8934130
s2.object_id   # => 8928350
```

```
s1 = "peanutbutter"
s2 = "peanut" + "butter"

s1.object_id   # => 8934130
s2.object_id   # => 8928350

sym1 = s1.to_sym
sym2 = s2.to_sym

sym1.object_id   # => 301218
sym2.object_id   # => 301218
```



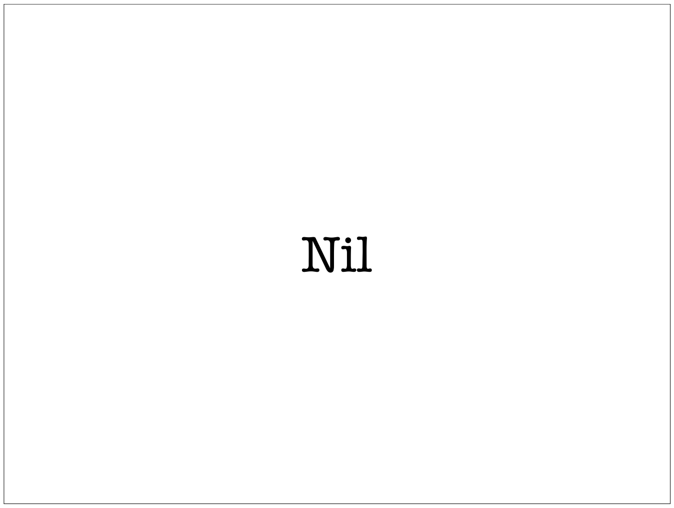
```
s1 = "peanutbutter"
s2 = "peanut" + "butter"

s1.object_id   # => 8934130
s2.object_id   # => 8928350

sym1 = s1.to_sym
sym2 = s2.to_sym

sym1.object_id   # => 301218
sym2.object_id   # => 301218
:peanutbutter.object_id
                 # => 301218
```

61



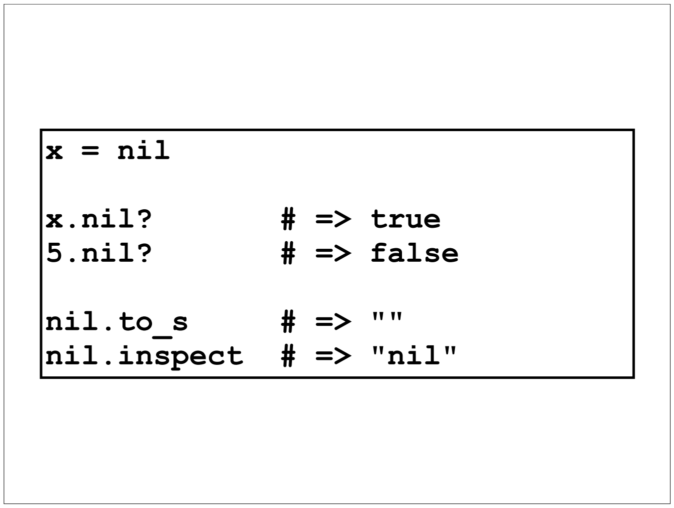
Used to **Name** Things

62



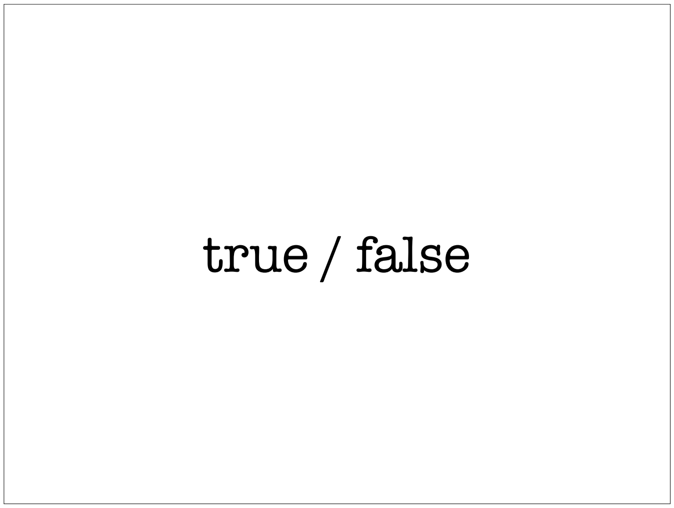
Nil

63



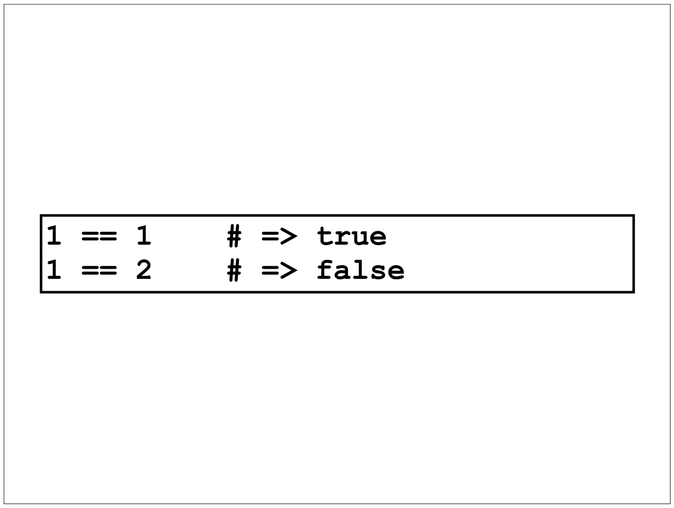
```
x = nil

x.nil?       # => true
5.nil?       # => false

nil.to_s     # => ""
nil.inspect  # => "nil"
```

64



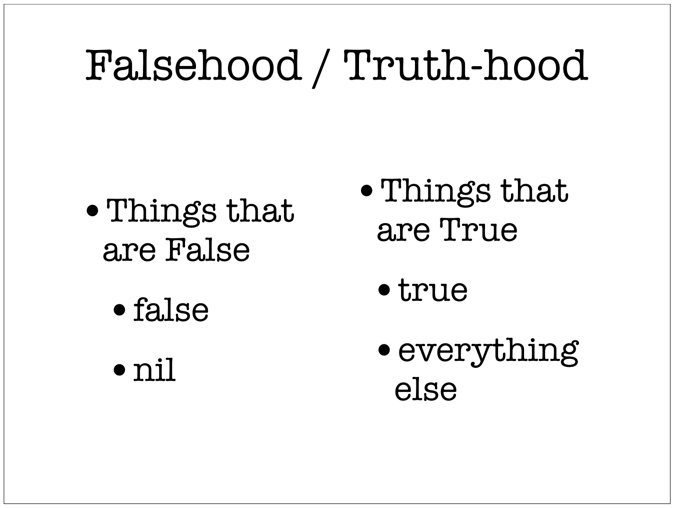
true / false

65



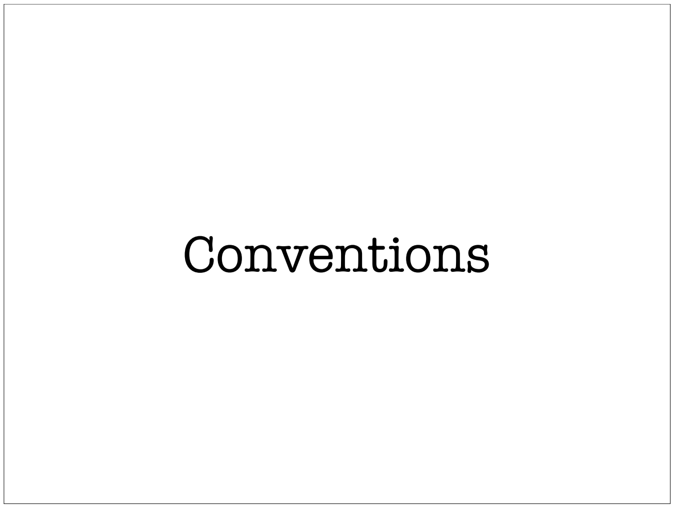
```
1 == 1    # => true
1 == 2    # => false
```

66

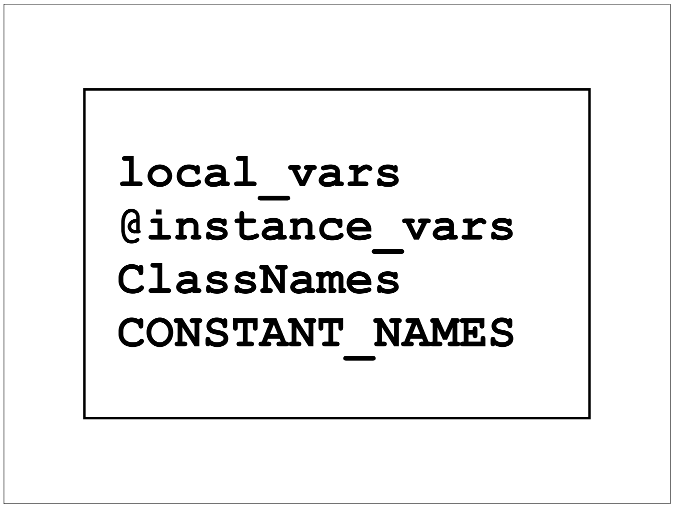


Falsehood / Truth-hood

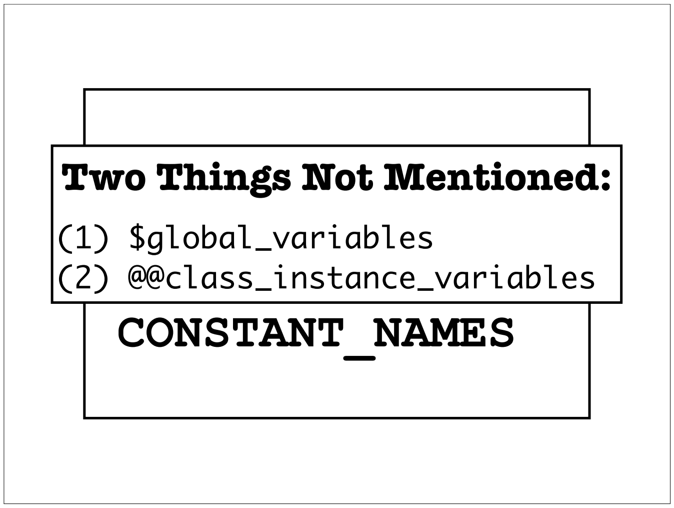
- Things that are False
  - false
  - nil
- Things that are True
  - true
  - everything else



Conventions



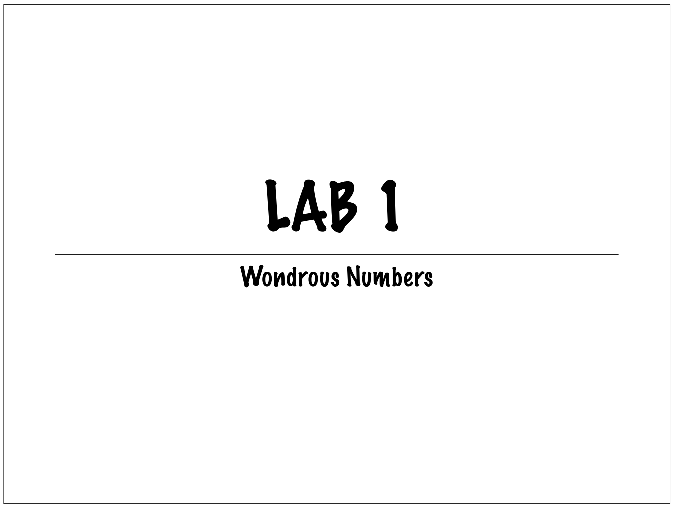
```
local_vars
@instance_vars
ClassNames
CONSTANT_NAMES
```



Two Things Not Mentioned:

```
(1) $global_variables
(2) @@class_instance_variables
```

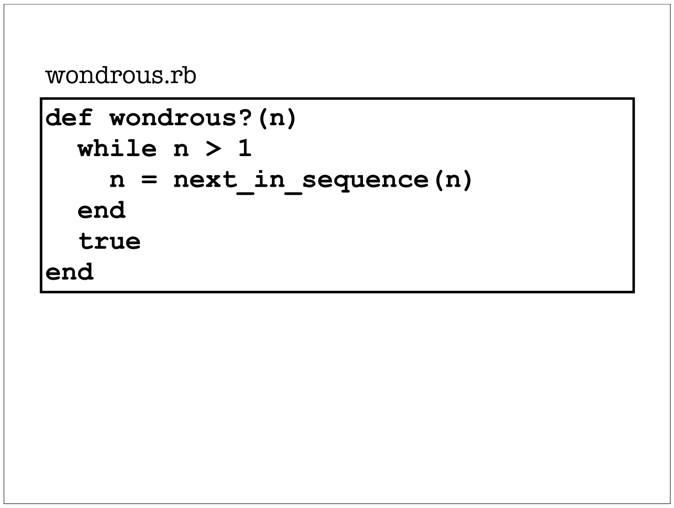
```
CONSTANT_NAMES
```



LAB 1

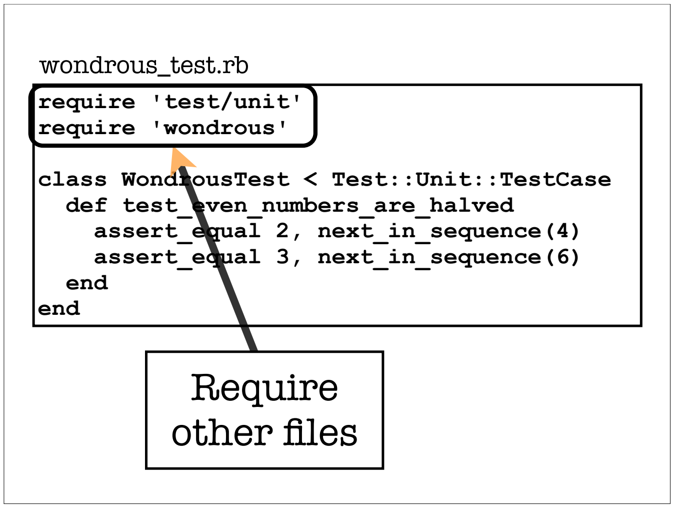
Wondrous Numbers

Testing



wondrous.rb

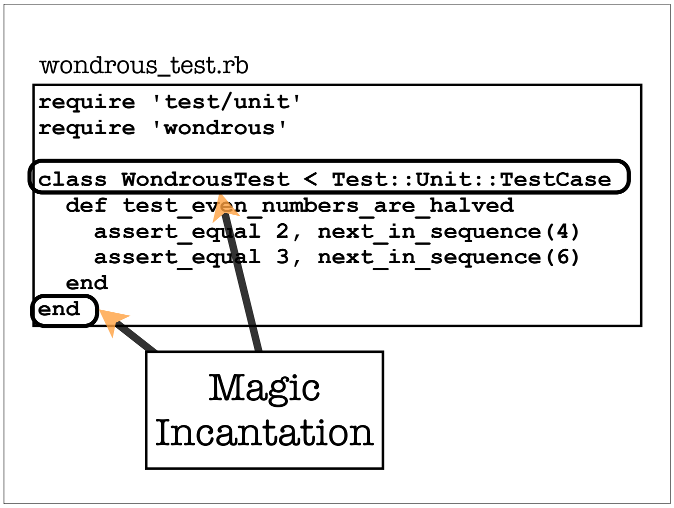
```
def wondrous?(n)
  while n > 1
    n = next_in_sequence(n)
  end
  true
end
```



wondrous_test.rb

```
require 'test/unit'
require 'wondrous'

class WondrousTest < Test::Unit::TestCase
  def test_even_numbers_are_halved
    assert_equal 2, next_in_sequence(4)
    assert_equal 3, next_in_sequence(6)
  end
end
```

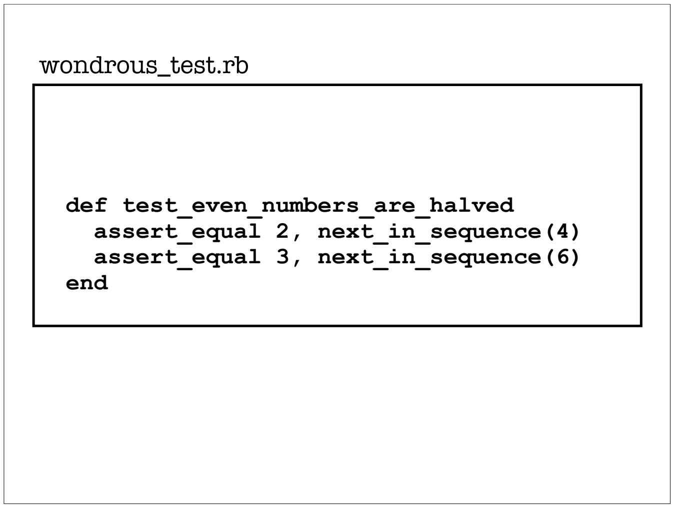
Require other files



wondrous_test.rb

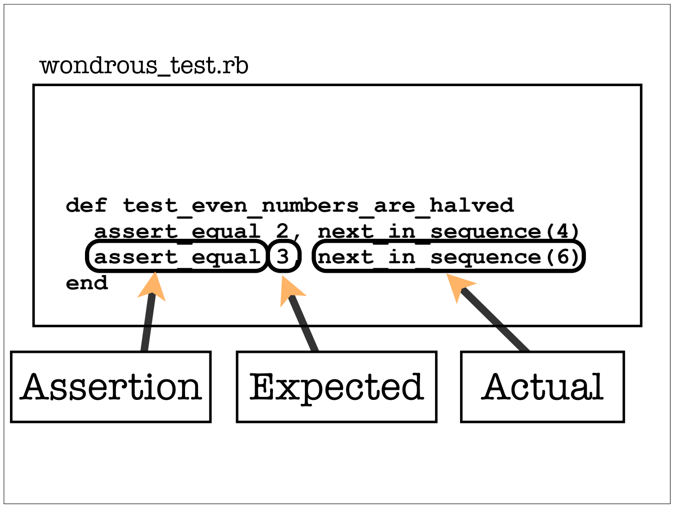
```
require 'test/unit'
require 'wondrous'

class WondrousTest < Test::Unit::TestCase
  def test_even_numbers_are_halved
    assert_equal 2, next_in_sequence(4)
    assert_equal 3, next_in_sequence(6)
  end
end
```

Magic Incantation



wondrous_test.rb

```
  def test_even_numbers_are_halved
    assert_equal 2, next_in_sequence(4)
    assert_equal 3, next_in_sequence(6)
  end
```



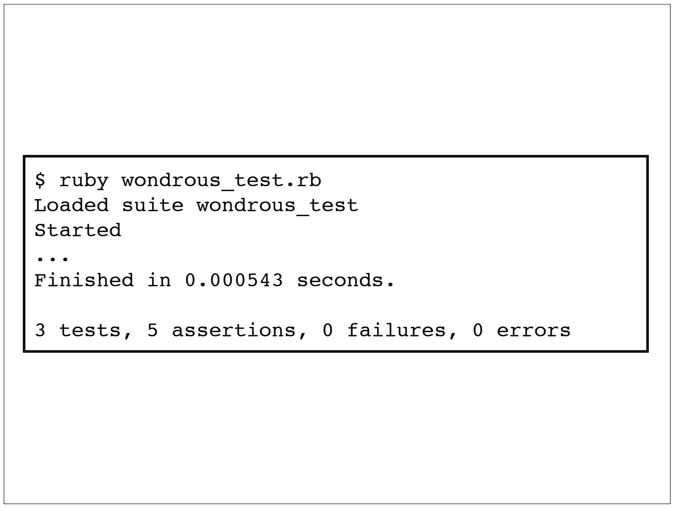
wondrous_test.rb

```
  def test_even_numbers_are_halved
    assert_equal 2, next_in_sequence(4)
    assert_equal 3, next_in_sequence(6)
  end
```

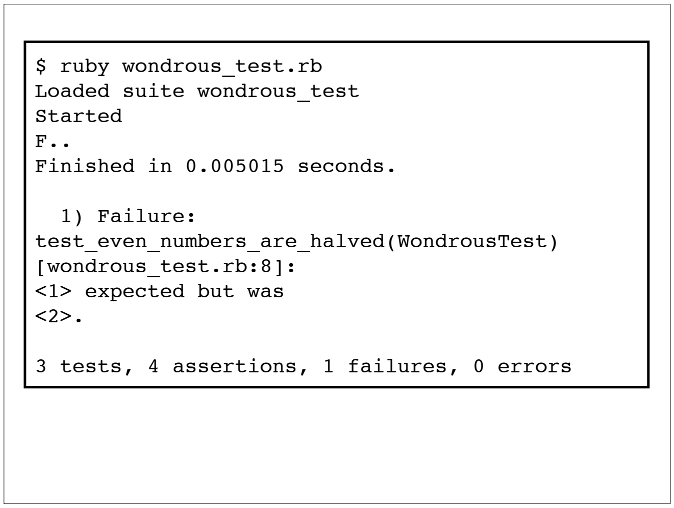
Assertion

Expected

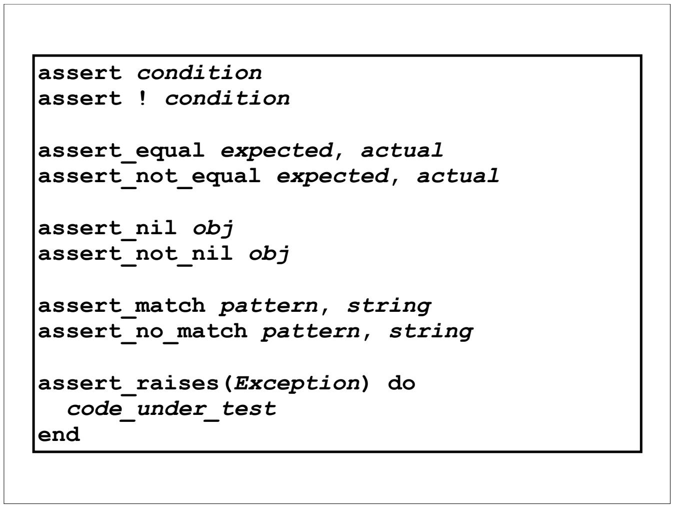
Actual



```
$ ruby wondrous_test.rb
Loaded suite wondrous_test
Started
...
Finished in 0.000543 seconds.

3 tests, 5 assertions, 0 failures, 0 errors
```

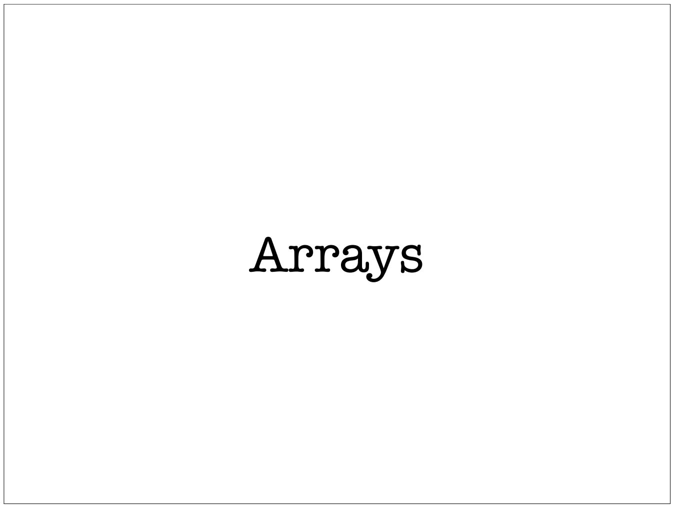


```
$ ruby wondrous_test.rb
Loaded suite wondrous_test
Started
F..
Finished in 0.005015 seconds.

  1) Failure:
test_even_numbers_are_halved(WondrousTest)
[wondrous_test.rb:8]:
<1> expected but was
<2>.

3 tests, 4 assertions, 1 failures, 0 errors
```

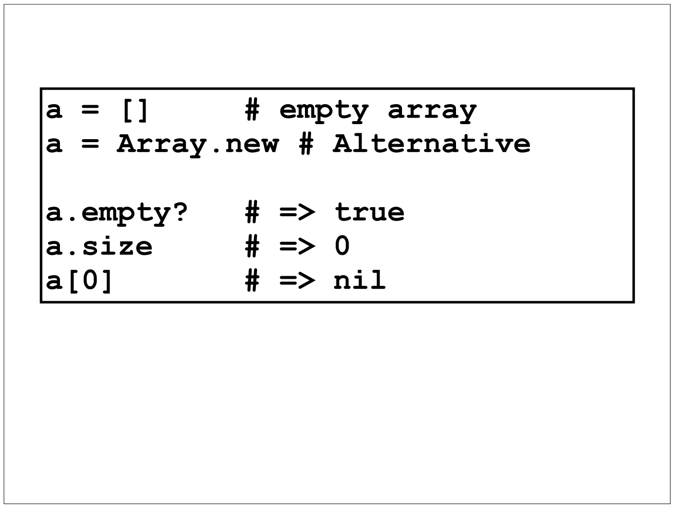


```
assert condition
assert ! condition

assert_equal expected, actual
assert_not_equal expected, actual

assert_nil obj
assert_not_nil obj

assert_match pattern, string
assert_no_match pattern, string

assert_raises(Exception) do
  code_under_test
end
```

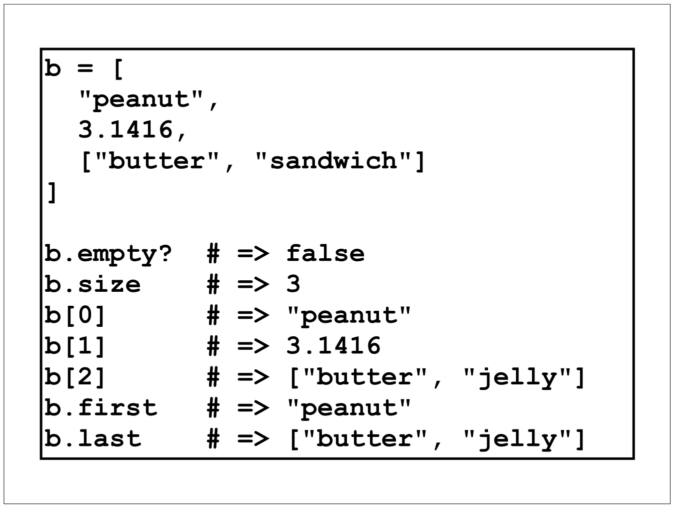
Containers



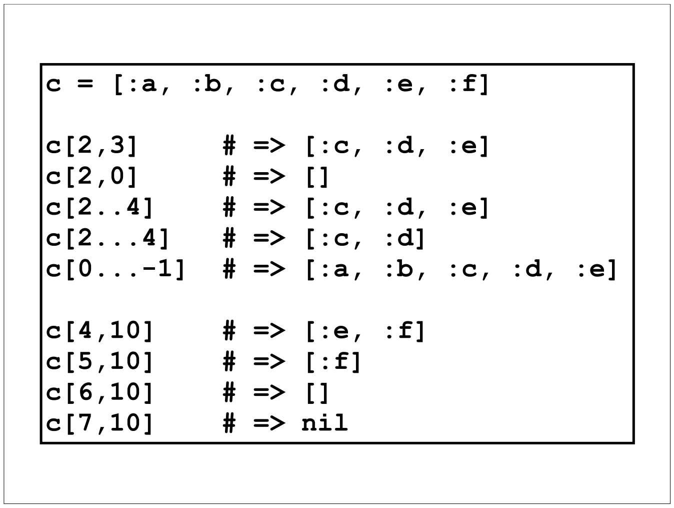
Arrays



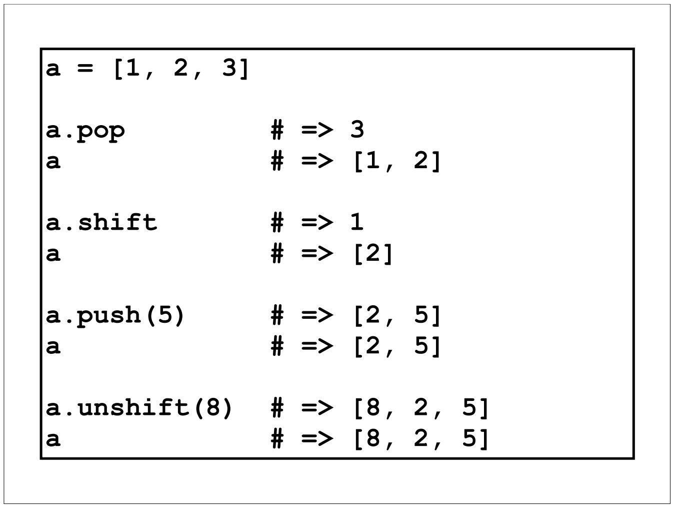
```
a = []      # empty array
a = Array.new # Alternative

a.empty?    # => true
a.size      # => 0
a[0]        # => nil
```



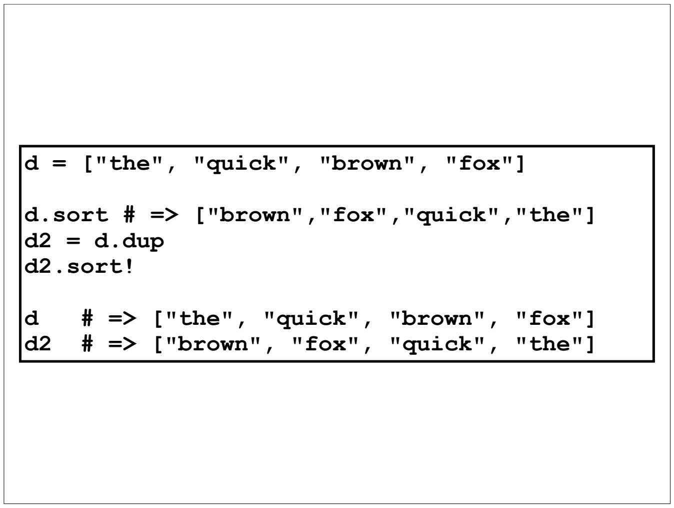
```
b = [
  "peanut",
  3.1416,
  ["butter", "sandwich"]
]

b.empty?  # => false
b.size    # => 3
b[0]      # => "peanut"
b[1]      # => 3.1416
b[2]      # => ["butter", "jelly"]
b.first   # => "peanut"
b.last    # => ["butter", "jelly"]
```



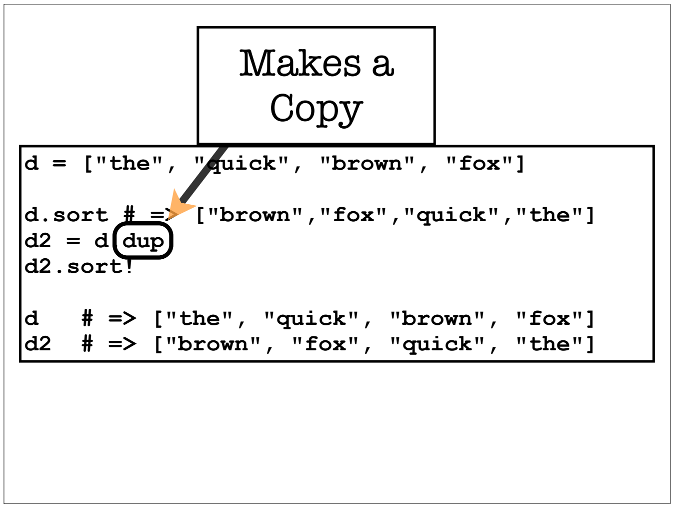
```
c = [:a, :b, :c, :d, :e, :f]

c[2,3]     # => [:c, :d, :e]
c[2,0]     # => []
c[2..4]    # => [:c, :d, :e]
c[2...4]   # => [:c, :d]
c[0...-1]  # => [:a, :b, :c, :d, :e]

c[4,10]    # => [:e, :f]
c[5,10]    # => [:f]
c[6,10]    # => []
c[7,10]    # => nil
```



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

a.pop          # => 3
a              # => [1, 2]

a.shift        # => 1
a              # => [2]

a.push(5)      # => [2, 5]
a              # => [2, 5]

a.unshift(8)   # => [8, 2, 5]
a              # => [8, 2, 5]
```

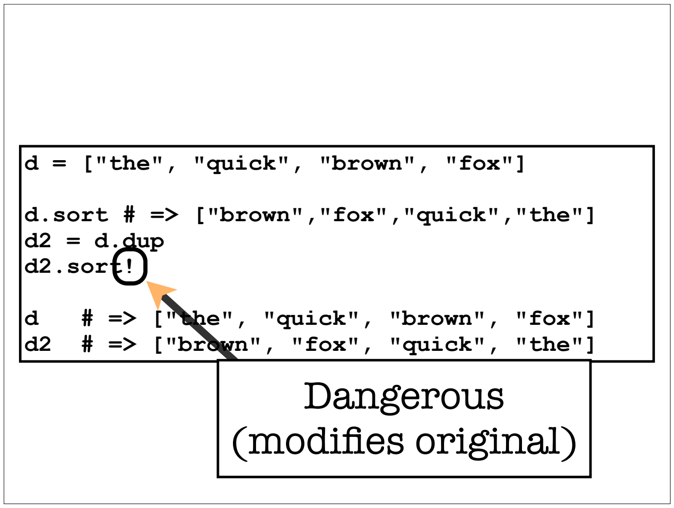


```
d = ["the", "quick", "brown", "fox"]

d.sort # => ["brown","fox","quick","the"]
d2 = d.dup
d2.sort!

d   # => ["the", "quick", "brown", "fox"]
d2  # => ["brown", "fox", "quick", "the"]
```



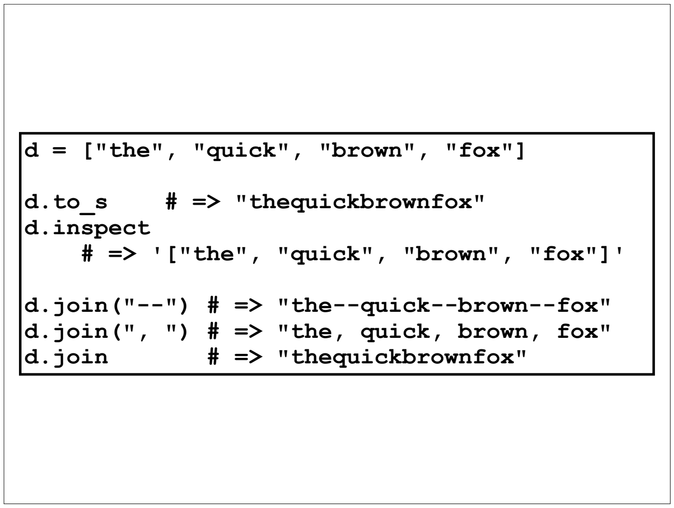
Makes a Copy

```
d = ["the", "quick", "brown", "fox"]

d.sort # => ["brown","fox","quick","the"]
d2 = d.dup
d2.sort!

d   # => ["the", "quick", "brown", "fox"]
d2  # => ["brown", "fox", "quick", "the"]
```



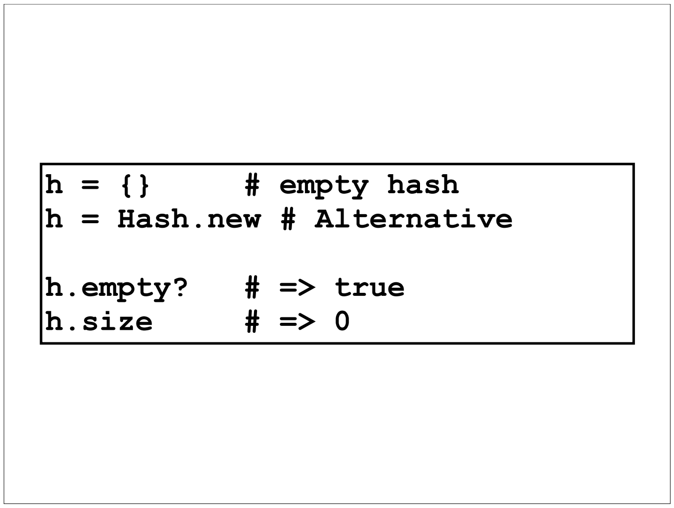
```
d = ["the", "quick", "brown", "fox"]

d.sort # => ["brown","fox","quick","the"]
d2 = d.dup
d2.sort!

d   # => ["the", "quick", "brown", "fox"]
d2  # => ["brown", "fox", "quick", "the"]
```

Dangerous (modifies original)

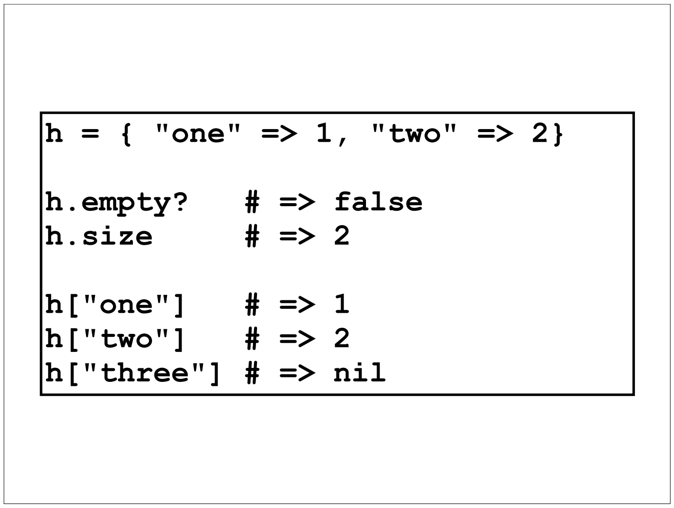


```
d = ["the", "quick", "brown", "fox"]

d.to_s    # => "thequickbrownfox"
d.inspect
    # => '["the", "quick", "brown", "fox"]'

d.join("--") # => "the--quick--brown--fox"
d.join(", ") # => "the, quick, brown, fox"
d.join       # => "thequickbrownfox"
```

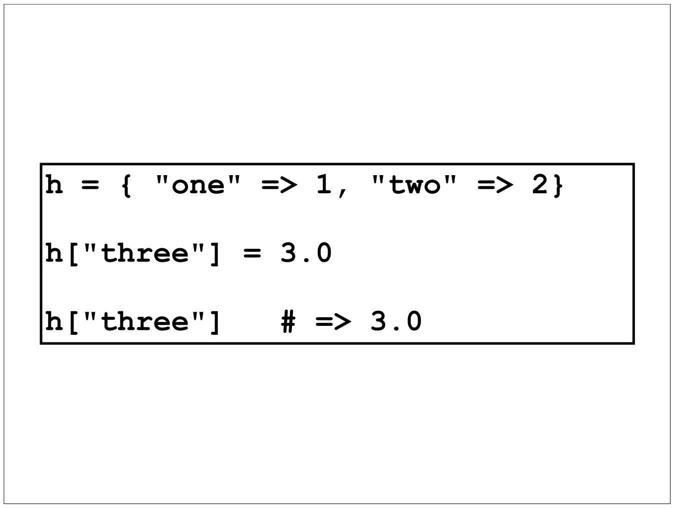
Hashes



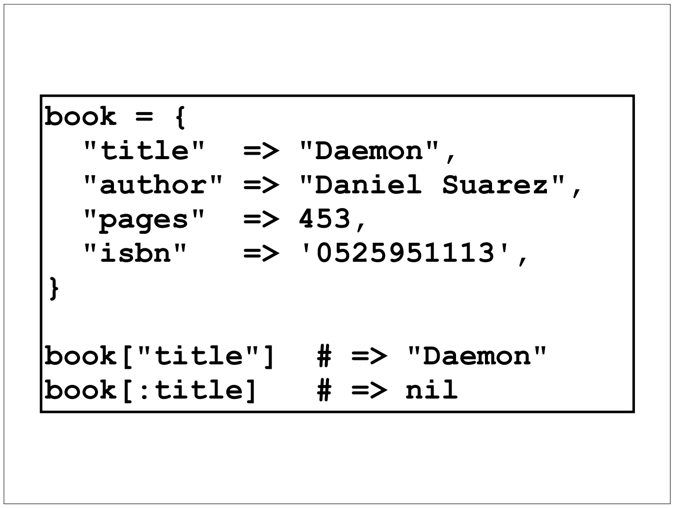
```
h = {}      # empty hash
h = Hash.new # Alternative

h.empty?   # => true
h.size     # => 0
```



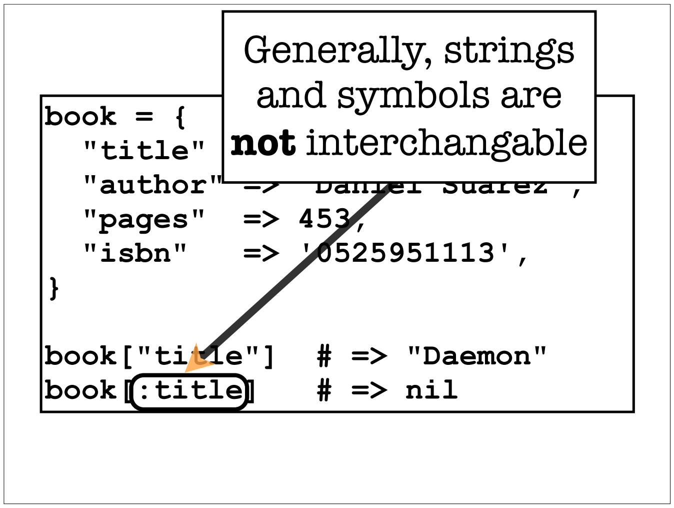
```
h = { "one" => 1, "two" => 2}

h.empty?   # => false
h.size     # => 2

h["one"]   # => 1
h["two"]   # => 2
h["three"] # => nil
```



```
h = { "one" => 1, "two" => 2}

h["three"] = 3.0

h["three"]   # => 3.0
```



```
book = {
  "title"  => "Daemon",
  "author" => "Daniel Suarez",
  "pages"  => 453,
  "isbn"   => '0525951113',
}

book["title"]  # => "Daemon"
book[:title]   # => nil
```



```
book = {
  "title"  => "Daemon",
  "author" => "Daniel Suarez",
  "pages"  => 453,
  "isbn"   => '0525951113',
}

book["title"]  # => "Daemon"
book[:title]   # => nil
```

Generally, strings and symbols are **not** interchangable

```
book.keys # => ["isbn", "title",
                "author", "pages]

book.values # => [
  '0525951113',
  "Daemon",
  "Daniel Suarez",
  448,
]
```

```
h = Hash.new
h[:key]    # => nil

h = Hash.new(100)
h[:key]    # => 100
```

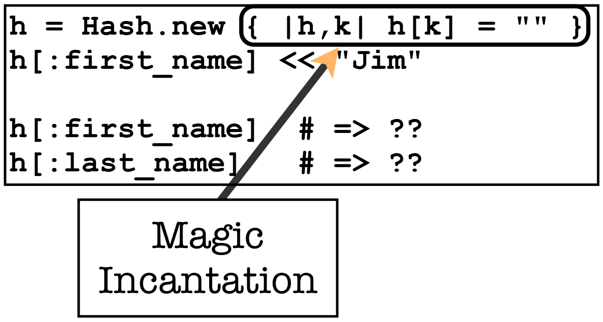
Try in IRB ...

```
h = Hash.new("")
h[:first_name] << "Jim"

h[:first_name]  # => ??
h[:last_name]   # => ??
```

Peeking Ahead

```
h = Hash.new { |h,k| h[k] = "" }
h[:first_name] << "Jim"

h[:first_name]  # => ??
h[:last_name]   # => ??
```

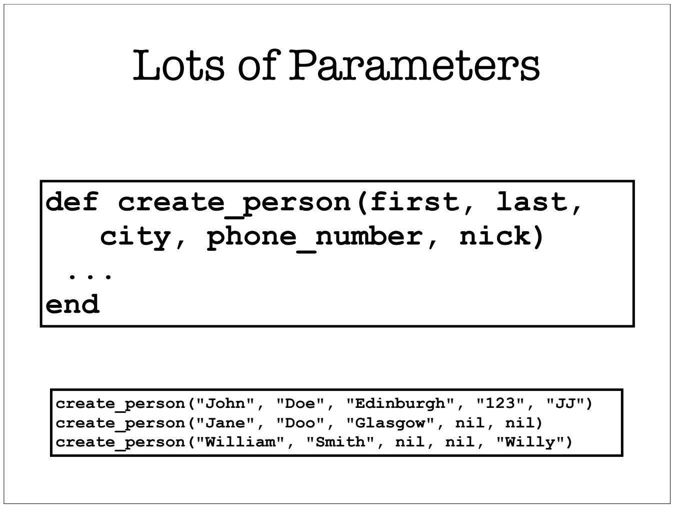
Peeking Ahead



```
h = Hash.new { |h,k| h[k] = "" }
h[:first_name] << "Jim"

h[:first_name]  # => ??
h[:last_name]   # => ??
```

Magic Incantation

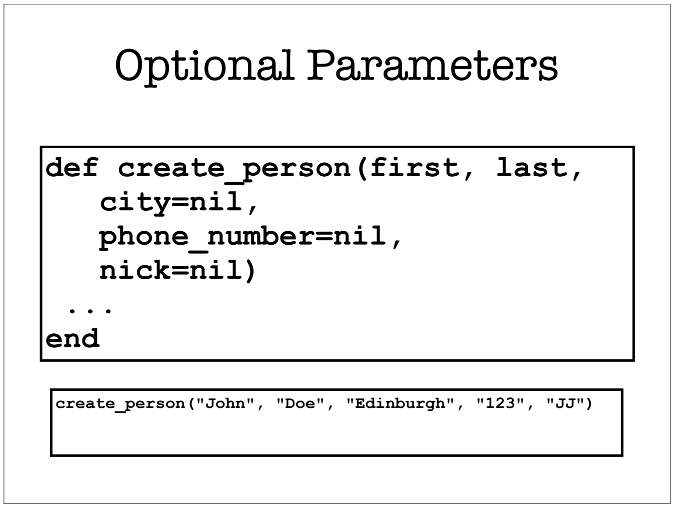
Hashes In Argument Lists



Lots of Parameters

```
def create_person(first, last,
   city, phone_number, nick)
 ...
end
```

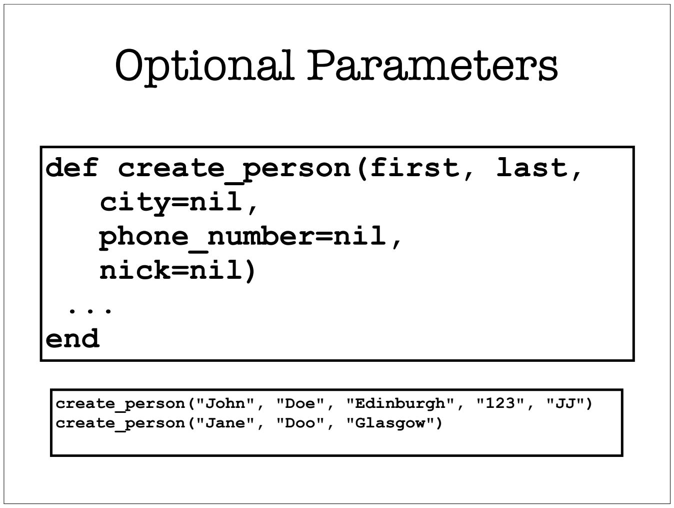
```
create_person("John", "Doe", "Edinburgh", "123", "JJ")
create_person("Jane", "Doo", "Glasgow", nil, nil)
create_person("William", "Smith", nil, nil, "Willy")
```



Optional Parameters

```
def create_person(first, last,
   city=nil,
   phone_number=nil,
   nick=nil)
 ...
end
```

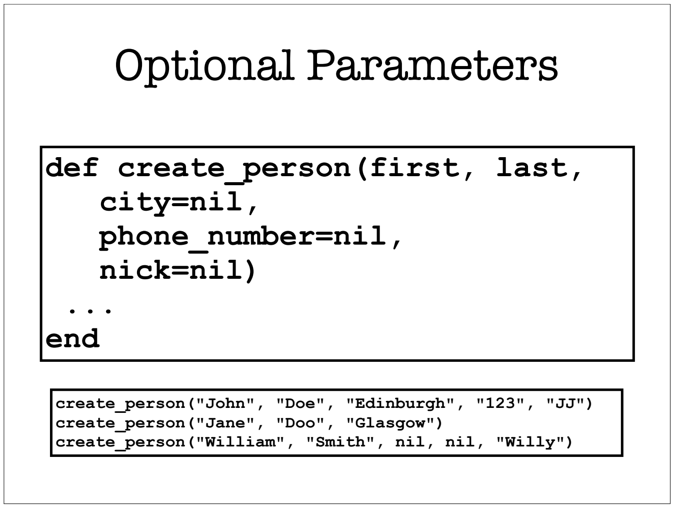
```
create_person("John", "Doe", "Edinburgh", "123", "JJ")
```



Optional Parameters

```
def create_person(first, last,
   city=nil,
   phone_number=nil,
   nick=nil)
 ...
end
```

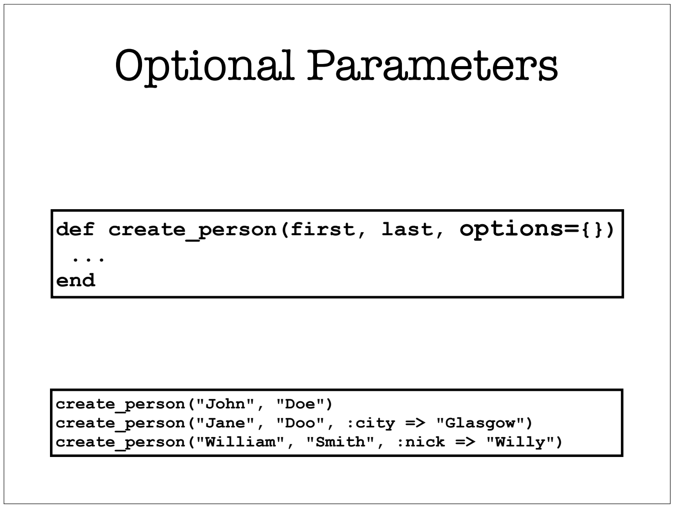
```
create_person("John", "Doe", "Edinburgh", "123", "JJ")
create_person("Jane", "Doo", "Glasgow")
```



Optional Parameters

```
def create_person(first, last,
   city=nil,
   phone_number=nil,
   nick=nil)
 ...
end
```

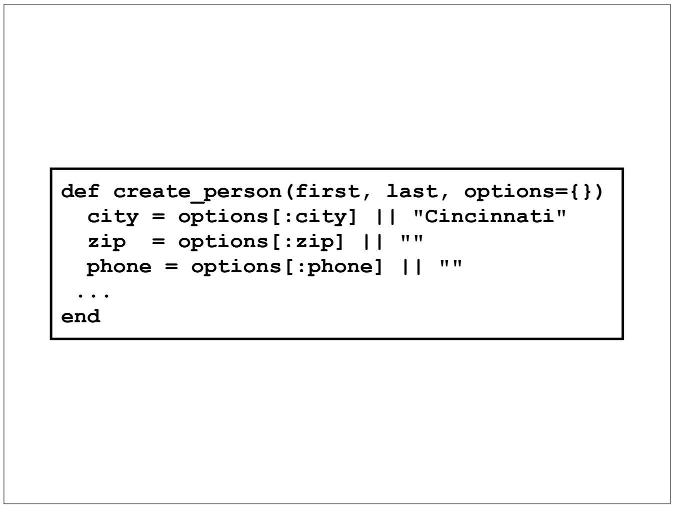
```
create_person("John", "Doe", "Edinburgh", "123", "JJ")
create_person("Jane", "Doo", "Glasgow")
create_person("William", "Smith", nil, nil, "Willy")
```



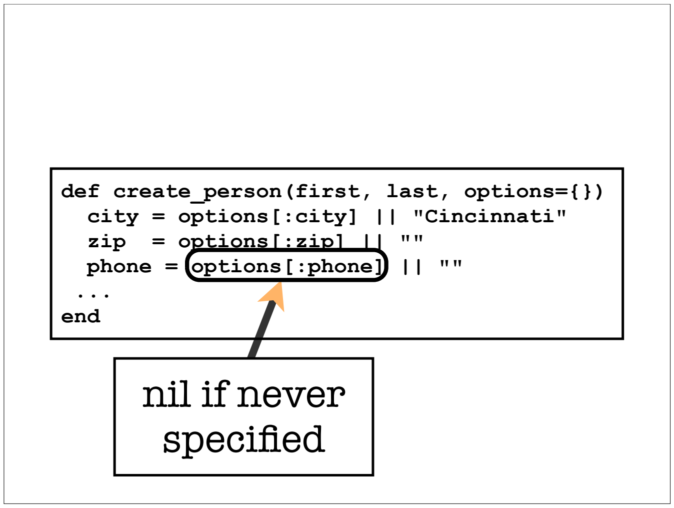
Optional Parameters

```
def create_person(first, last, options={})
 ...
end
```

```
create_person("John", "Doe")
create_person("Jane", "Doo", :city => "Glasgow")
create_person("William", "Smith", :nick => "Willy")
```

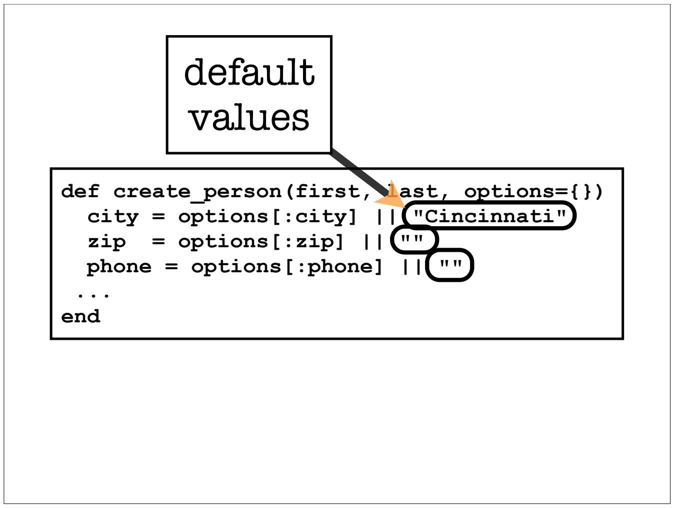


```
def create_person(first, last, options={})
  city = options[:city] || "Cincinnati"
  zip  = options[:zip] || ""
  phone = options[:phone] || ""
 ...
end
```



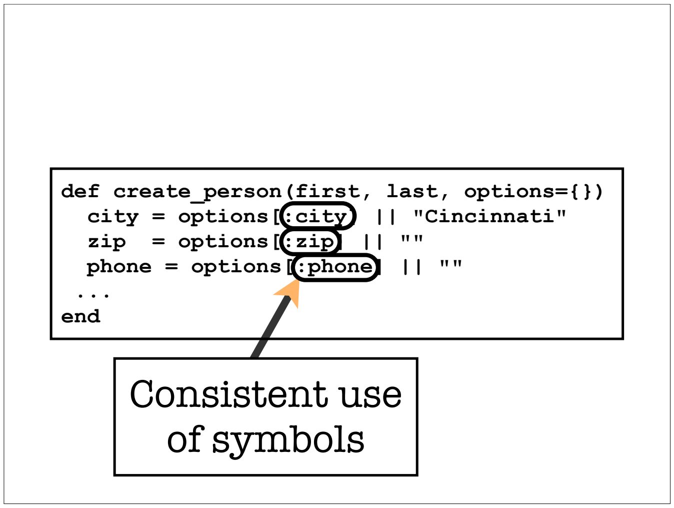
```
def create_person(first, last, options={})
  city = options[:city] || "Cincinnati"
  zip  = options[:zip] || ""
  phone = options[:phone] || ""
 ...
end
```

nil if never specified



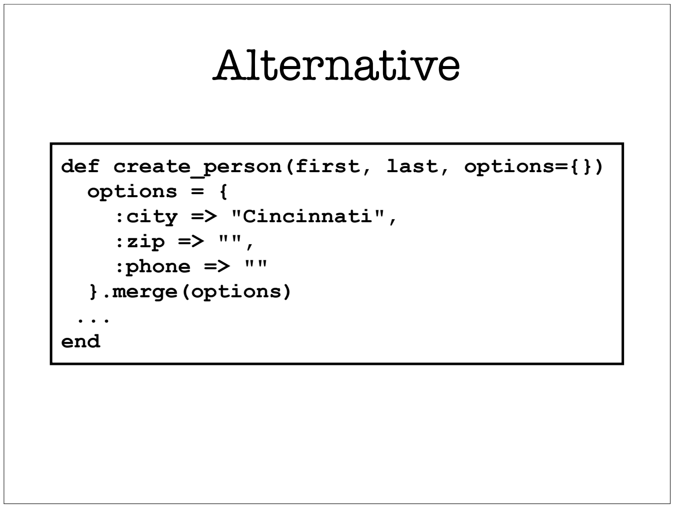
default values

```
def create_person(first, last, options={})
  city = options[:city] || "Cincinnati"
  zip  = options[:zip] || ""
  phone = options[:phone] || ""
 ...
end
```



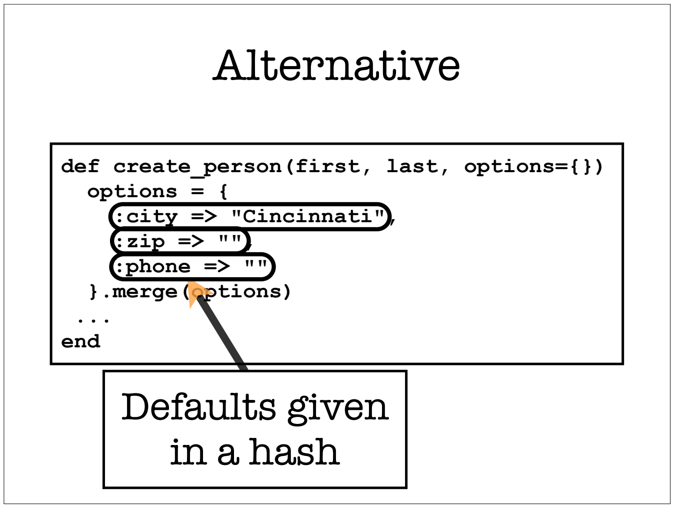
```
def create_person(first, last, options={})
  city = options[:city] || "Cincinnati"
  zip  = options[:zip] || ""
  phone = options[:phone] || ""
 ...
end
```

Consistent use of symbols



Alternative

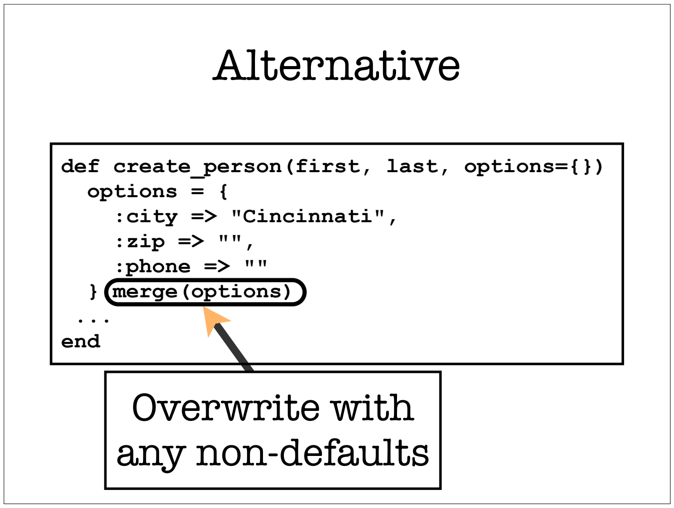
```
def create_person(first, last, options={})
  options = {
    :city => "Cincinnati",
    :zip => "",
    :phone => ""
  }.merge(options)
 ...
end
```



Alternative

```
def create_person(first, last, options={})
  options = {
    :city => "Cincinnati",
    :zip => "",
    :phone => ""
  }.merge(options)
 ...
end
```

Defaults given in a hash



Alternative

```
def create_person(first, last, options={})
  options = {
    :city => "Cincinnati",
    :zip => "",
    :phone => ""
  }.merge(options)
 ...
end
```

Overwrite with any non-defaults

While We're Talking about Method Arguments

Given

```
def f(a, b="B", *args)
  puts "a=#{a.inspect}"
  puts "b=#{b.inspect}"
  puts "args=#{args.inspect}"
end
```

What's the Output?

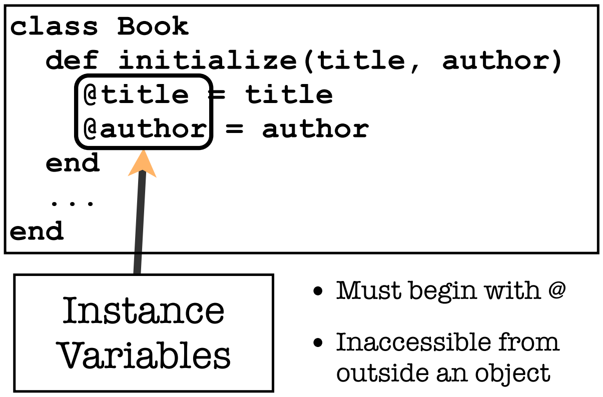
```
f("X")
f("X", "Y")
f("X", "Y", "Z")
f("X", "Y", "Z", "XYZZY")

args = [
  "one", "two", "three", "four"
]
f(*args)
```

LAB 2

Wondrous Sequences

Classes

```
class Book
  def initialize(title, author)
    @title = title
    @author = author
  end
  ...
end
```

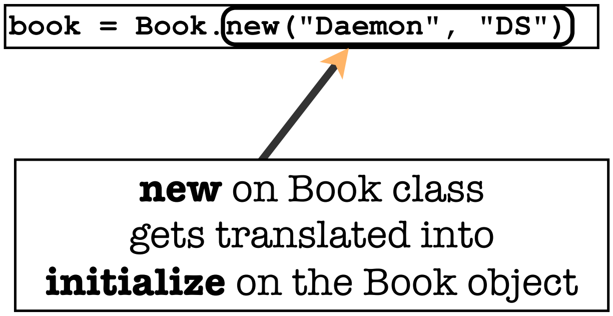


```
class Book
  def initialize(title, author)
    @title = title
    @author = author
  end
  ...
end
```

Instance Variables

- Must begin with @
- Inaccessible from outside an object

```
book = Book.new("Daemon", "DS")
```



```
book = Book.new("Daemon", "DS")
```

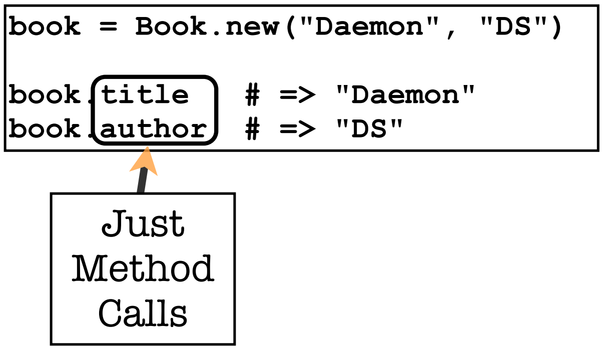
new on Book class gets translated into **initialize** on the Book object

```
book = Book.new("Daemon", "DS")

book.????
```

How can we get to the books author and title?

```
class Book
  def initialize(title, author)
    @title = title
    @author = author
  end
  def author
    @author
  end
  def title
    @title
  end
end
```

```
book = Book.new("Daemon", "DS")

book.title    # => "Daemon"
book.author   # => "DS"
```



```
book = Book.new("Daemon", "DS")

book.title   # => "Daemon"
book.author  # => "DS"
```

Just Method Calls

IMPORTANT!

The **only** way to talk to an object is by calling methods!

IMPORTANT!

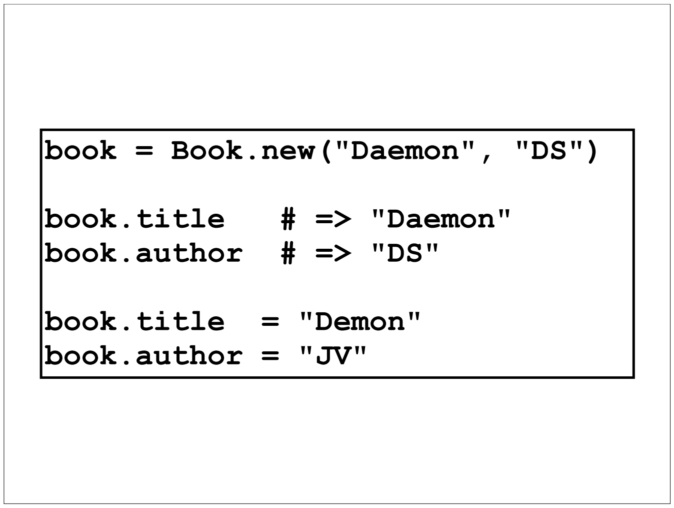
The **only** way to talk to an object is by calling methods!

(i.e. sending messages)

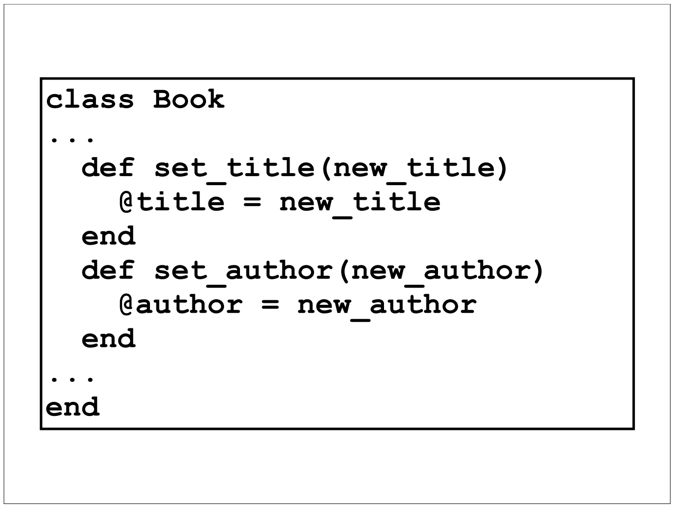
```
book = Book.new("Daemon", "DS")

book.title    # => "Daemon"
book.author   # => "DS"

book.set_title("Demon")
book.set_author("JV")
```

```
class Book
...
  def set_title(new_title)
    @title = new_title
  end
  def set_author(new_author)
    @author = new_author
  end
...
end
```

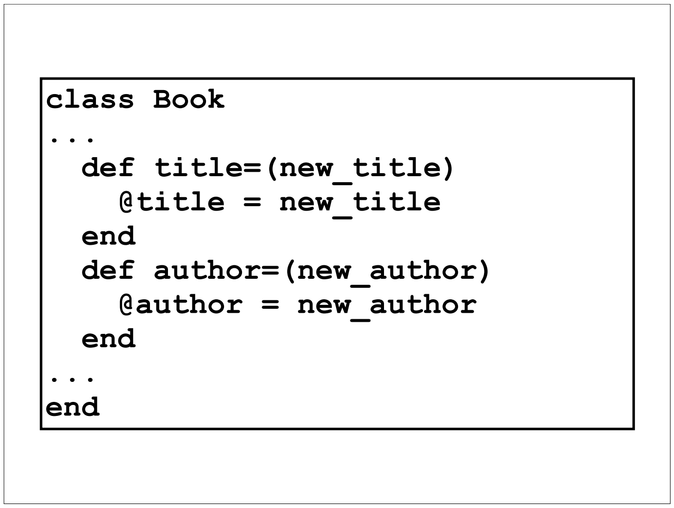
```
book = Book.new("Daemon", "DS")

book.title    # => "Daemon"
book.author   # => "DS"

book.set_title("Demon")
book.set_author("JV")
```



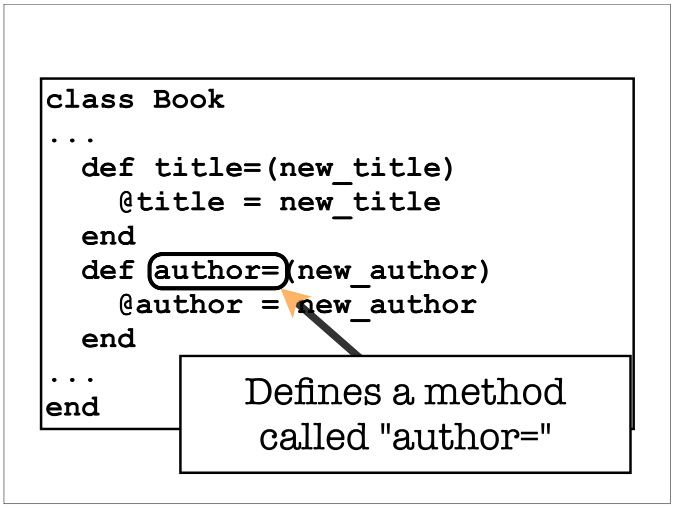
```
book = Book.new("Daemon", "DS")

book.title    # => "Daemon"
book.author   # => "DS"

book.title  = "Demon"
book.author = "JV"
```



```
class Book
...
  def set_title(new_title)
    @title = new_title
  end
  def set_author(new_author)
    @author = new_author
  end
...
end
```

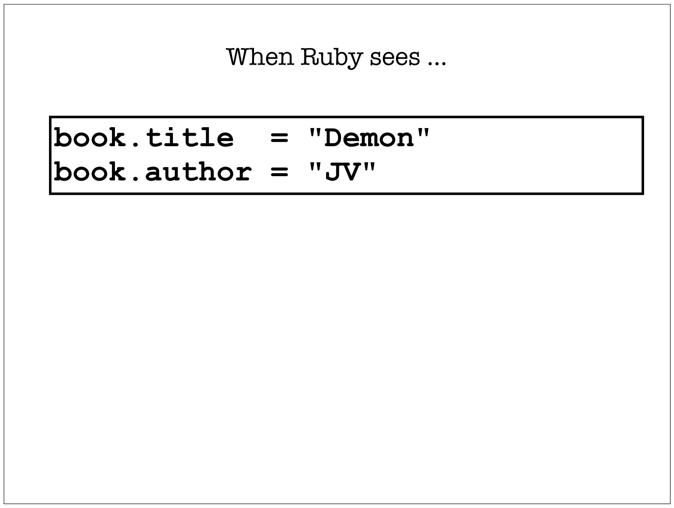


```
class Book
...
  def title=(new_title)
    @title = new_title
  end
  def author=(new_author)
    @author = new_author
  end
...
end
```



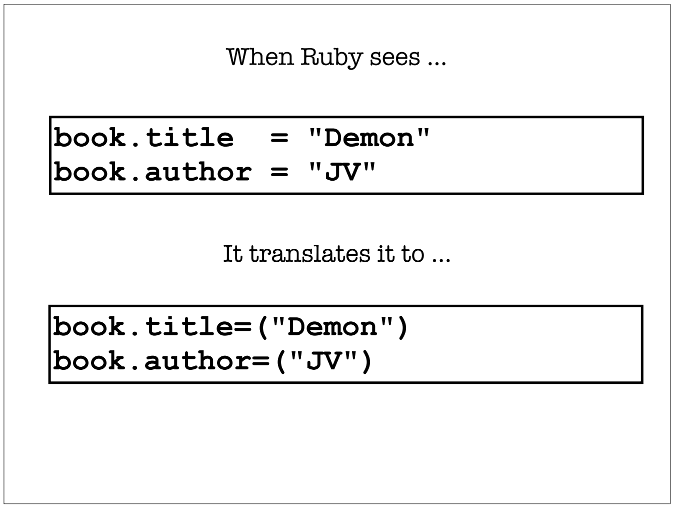
```
class Book
...
  def title=(new_title)
    @title = new_title
  end
  def author=(new_author)
    @author = new_author
  end
...
end
```

Defines a method called "author="



When Ruby sees ...

```
book.title  = "Demon"
book.author = "JV"
```

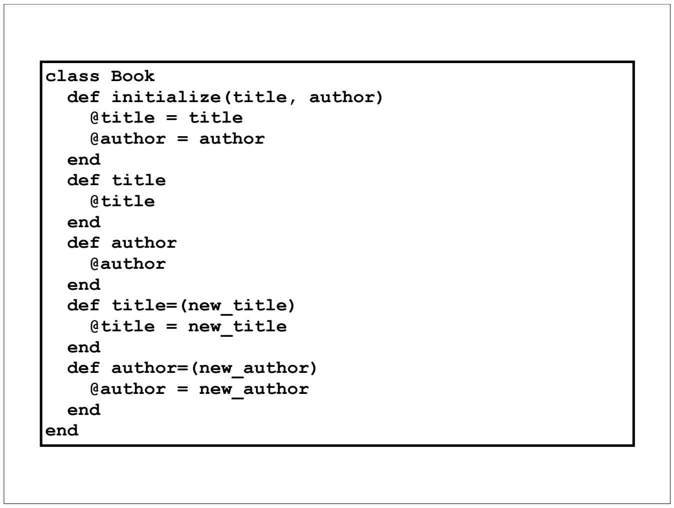


When Ruby sees ...

```
book.title  = "Demon"
book.author = "JV"
```

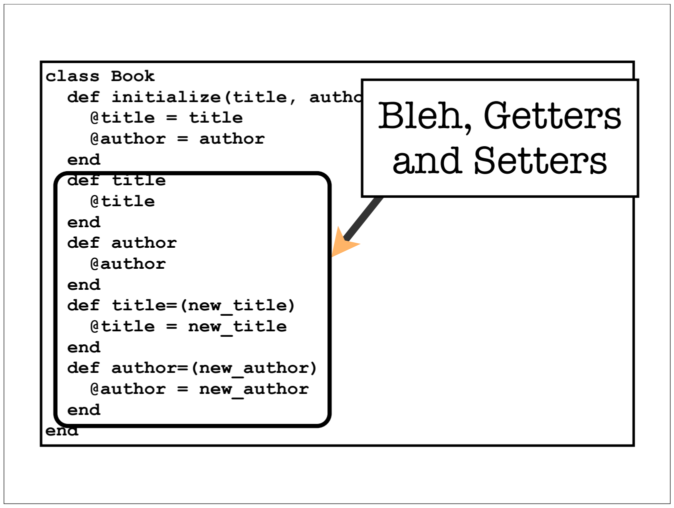
It translates it to ...

```
book.title=("Demon")
book.author=("JV")
```



```
class Book
  def initialize(title, author)
    @title = title
    @author = author
  end
  def title
    @title
  end
  def author
    @author
  end
  def title=(new_title)
    @title = new_title
  end
  def author=(new_author)
    @author = new_author
  end
end
```

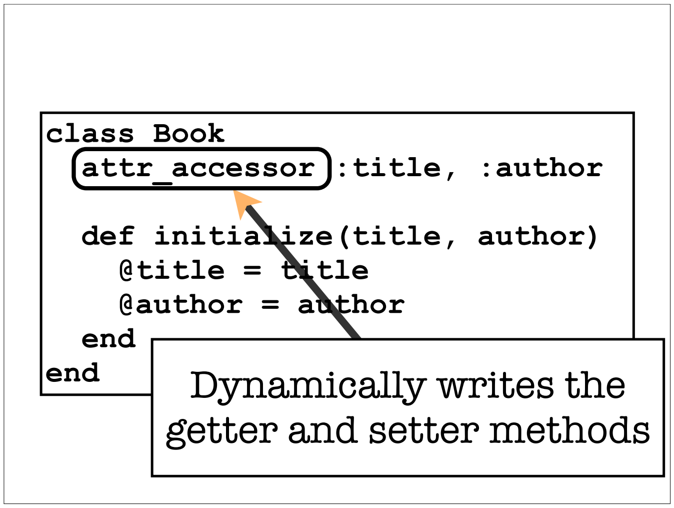
139



```
class Book
  def initialize(title, autho
    @title = title
    @author = author
  end
  def title
    @title
  end
  def author
    @author
  end
  def title=(new_title)
    @title = new_title
  end
  def author=(new_author)
    @author = new_author
  end
end
```

Bleh, Getters and Setters

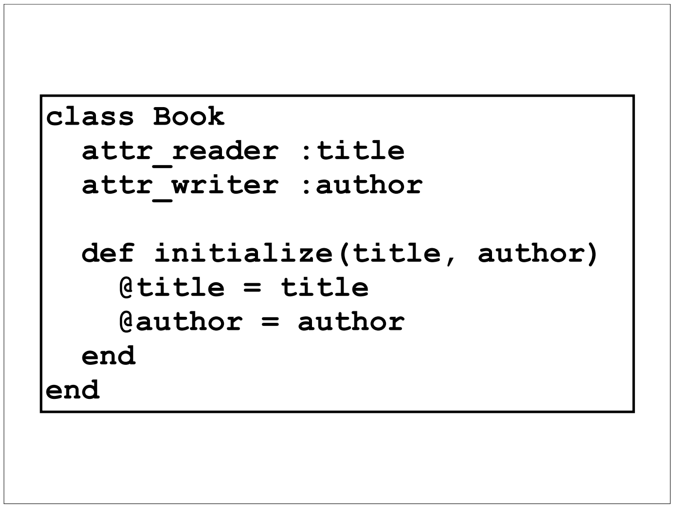
140



```
class Book
  attr_accessor :title, :author

  def initialize(title, author)
    @title = title
    @author = author
  end
end
```

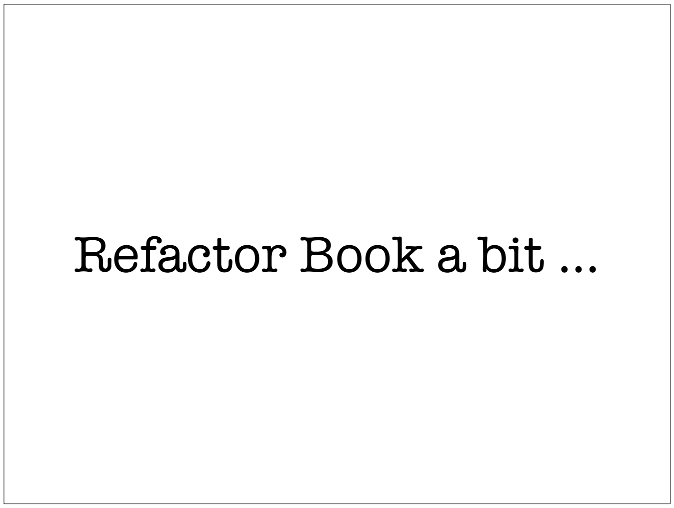
Dynamically writes the getter and setter methods

141



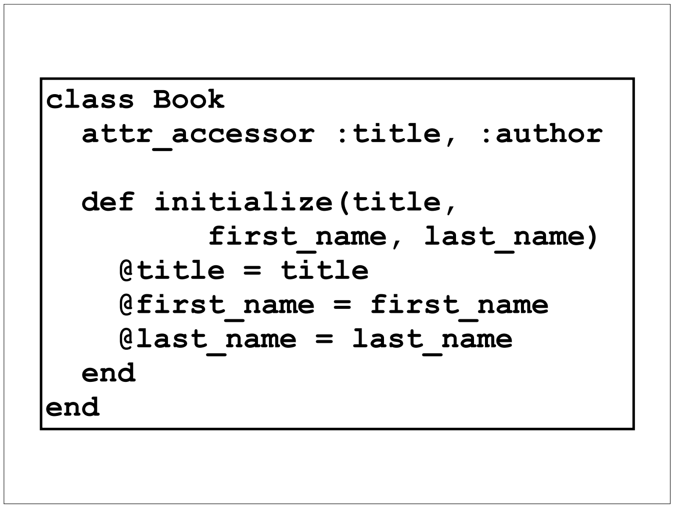
```
class Book
  attr_reader :title
  attr_writer :author

  def initialize(title, author)
    @title = title
    @author = author
  end
end
```

142



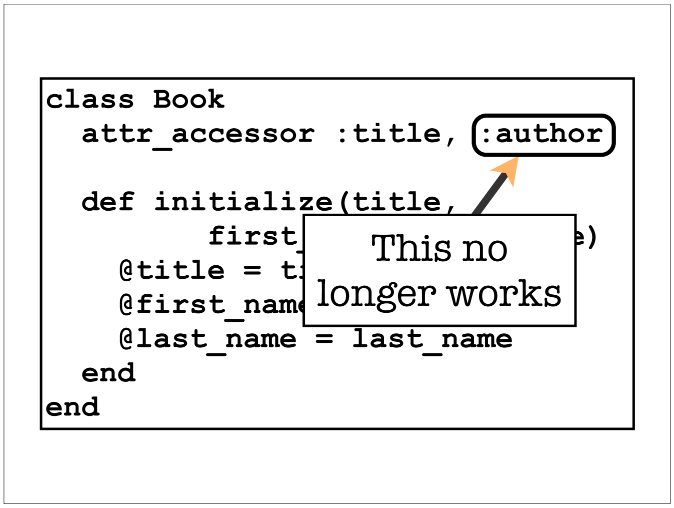
Refactor Book a bit ...

143



```
class Book
  attr_accessor :title, :author

  def initialize(title,
         first_name, last_name)
    @title = title
    @first_name = first_name
    @last_name = last_name
  end
end
```

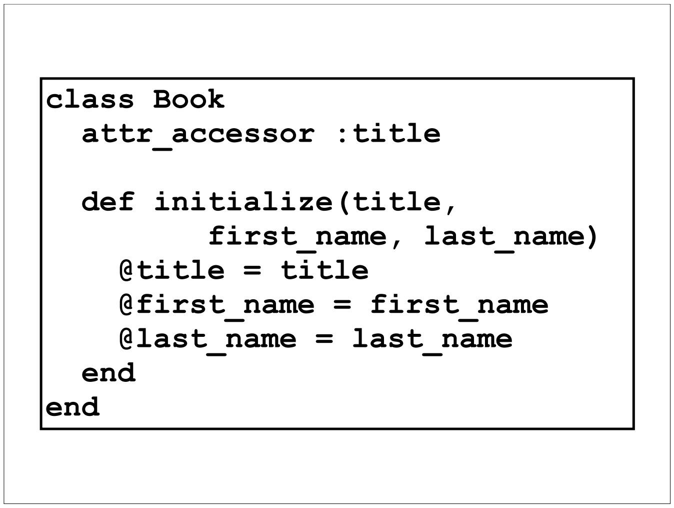
144



```
class Book
  attr_accessor :title, :author

  def initialize(title,
         first_                e)
    @title = ti
    @first_name
    @last_name = last_name
  end
end
```

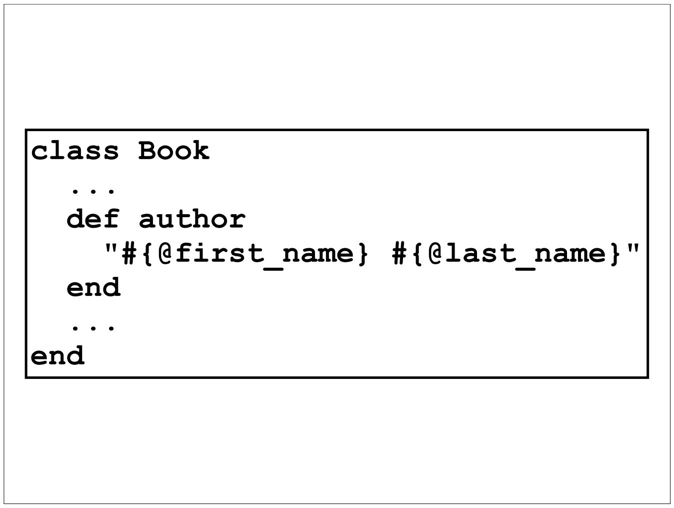
This no longer works

145



```
class Book
  attr_accessor :title

  def initialize(title,
         first_name, last_name)
    @title = title
    @first_name = first_name
    @last_name = last_name
  end
end
```

146

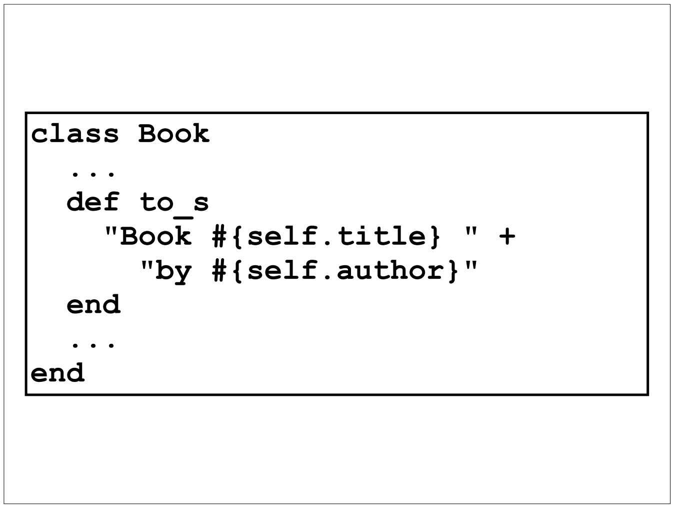


```
class Book
  ...
  def author
    "#{@first_name} #{@last_name}"
  end
  ...
end
```

147

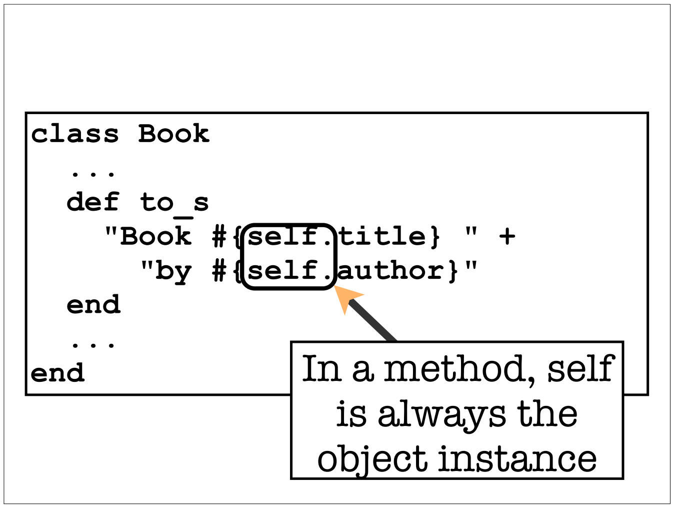
self

148



```
class Book
  ...
  def to_s
    "Book #{self.title} " +
      "by #{self.author}"
  end
  ...
end
```

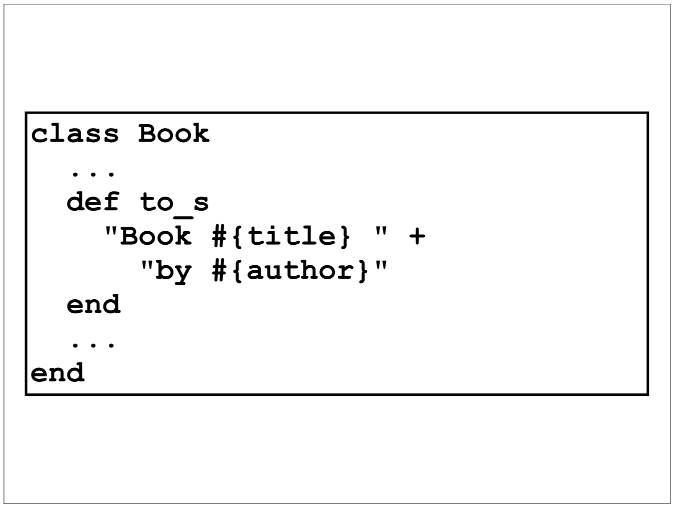
149



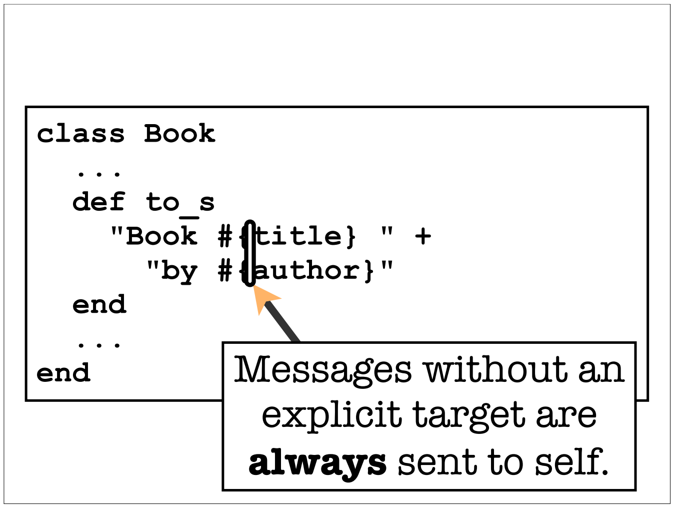
```
class Book
  ...
  def to_s
    "Book #{self.title} " +
      "by #{self.author}"
  end
  ...
end
```

In a method, self is always the object instance

150

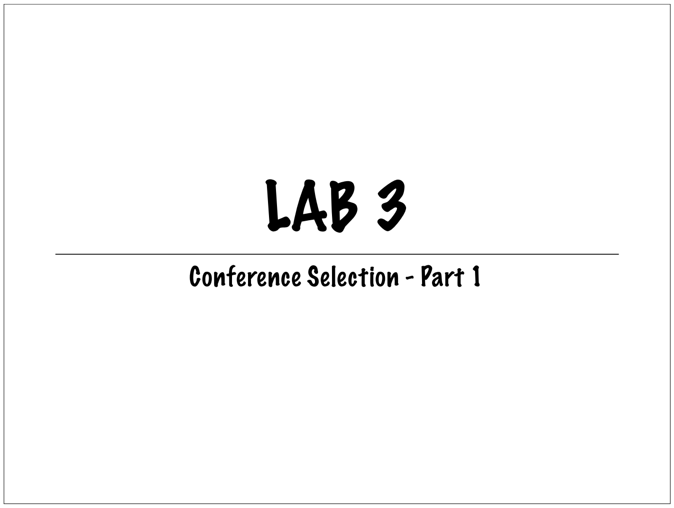


```
class Book
  ...
  def to_s
    "Book #{title} " +
      "by #{author}"
  end
  ...
end
```



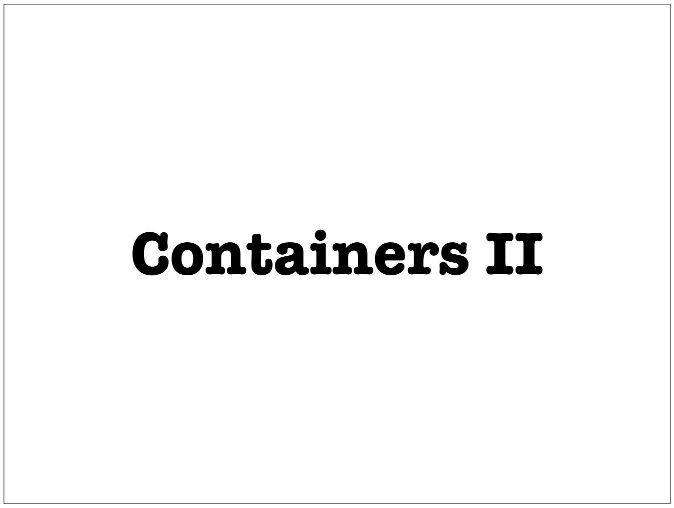
```
class Book
  ...
  def to_s
    "Book #{title} " +
      "by #{author}"
  end
  ...
end
```

Messages without an explicit target are **always** sent to self.

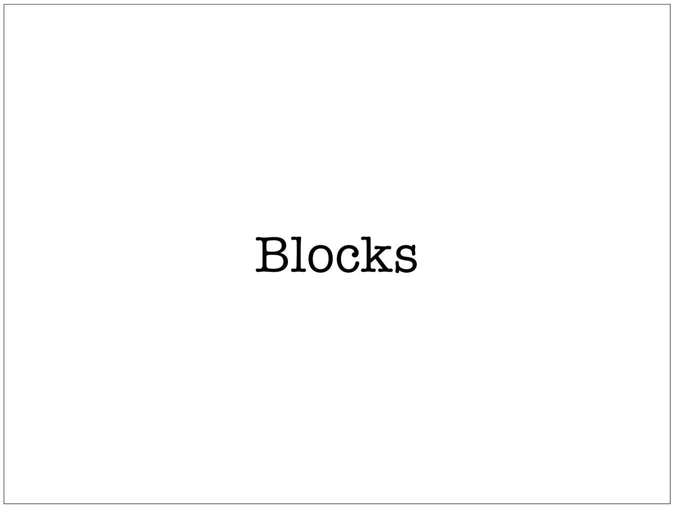


LAB 3

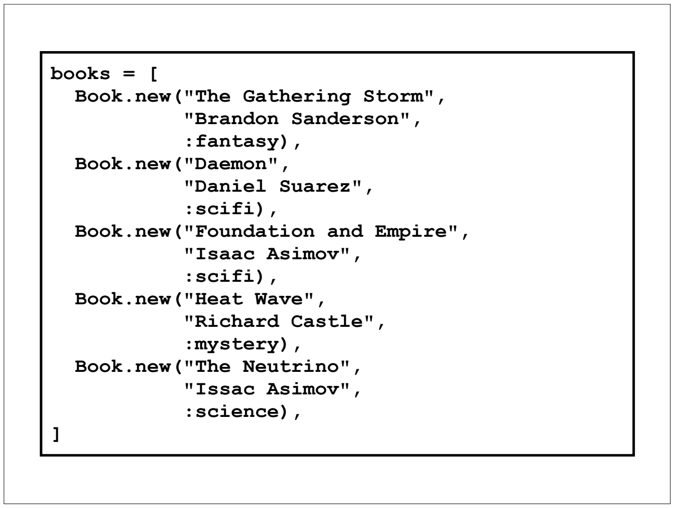
Conference Selection - Part 1



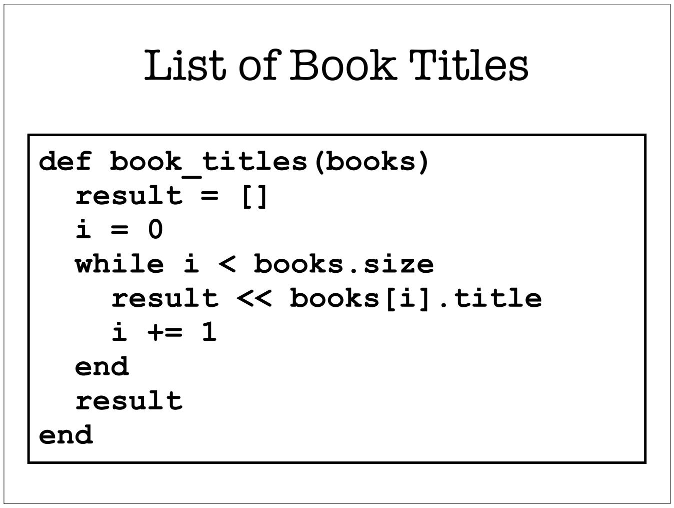
Containers II



Blocks



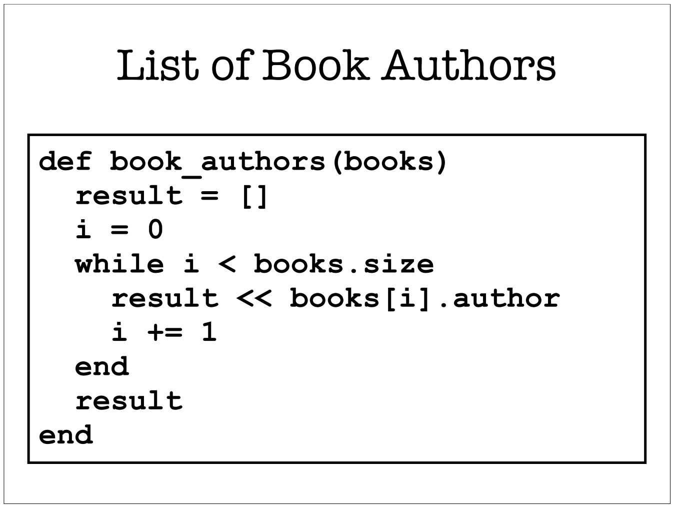
```
books = [
  Book.new("The Gathering Storm",
           "Brandon Sanderson",
           :fantasy),
  Book.new("Daemon",
           "Daniel Suarez",
           :scifi),
  Book.new("Foundation and Empire",
           "Isaac Asimov",
           :scifi),
  Book.new("Heat Wave",
           "Richard Castle",
           :mystery),
  Book.new("The Neutrino",
           "Issac Asimov",
           :science),
]
```



List of Book Titles

```
def book_titles(books)
  result = []
  i = 0
  while i < books.size
    result << books[i].title
    i += 1
  end
  result
end
```

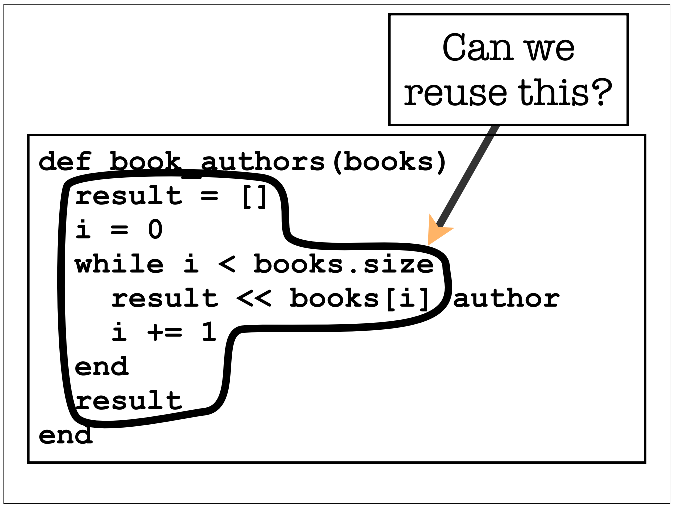
157



List of Book Authors

```
def book_authors(books)
  result = []
  i = 0
  while i < books.size
    result << books[i].author
    i += 1
  end
  result
end
```

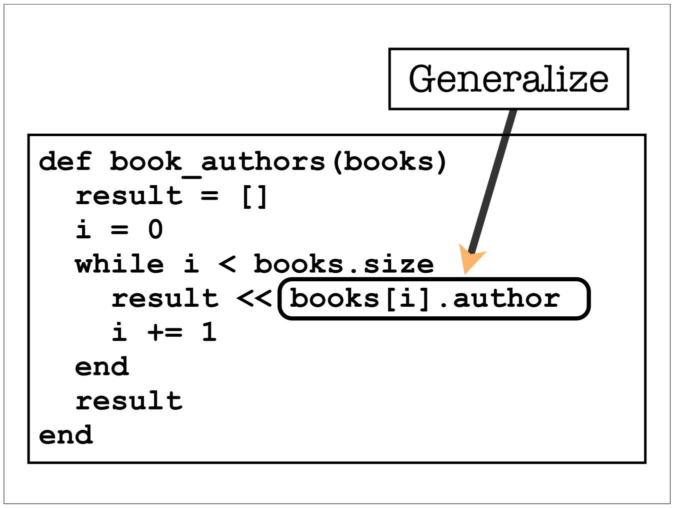
158



Can we reuse this?

```
def book_authors(books)
  result = []
  i = 0
  while i < books.size
    result << books[i].author
    i += 1
  end
  result
end
```

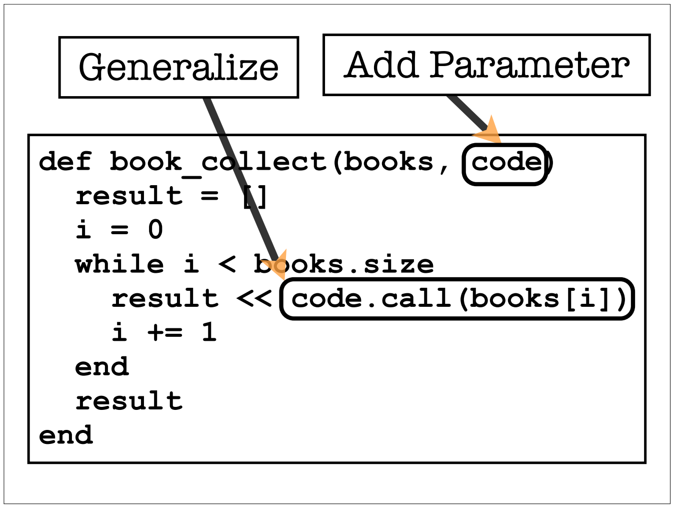
159



Generalize

```
def book_authors(books)
  result = []
  i = 0
  while i < books.size
    result << books[i].author
    i += 1
  end
  result
end
```

160

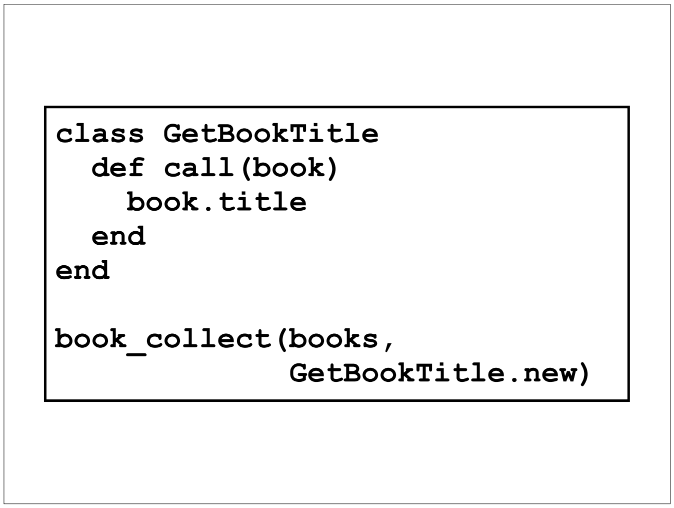


Generalize

Add Parameter

```
def book_collect(books, code)
  result = []
  i = 0
  while i < books.size
    result << code.call(books[i])
    i += 1
  end
  result
end
```

161



```
class GetBookTitle
  def call(book)
    book.title
  end
end

book_collect(books,
             GetBookTitle.new)
```

162

```
class GetBookAuthor
  def call(book)
    book.author
  end
end

book_collect(books,
             GetBookAuthor.new)
```

No Book-Specific Code

```
def book_collect(books, code)
  result = []
  i = 0
  while i < books.size
    result << code.call(books[i])
    i += 1
  end
  result
end
```

Remove Book References

```
def collect(items, code)
  result = []
  i = 0
  while i < items.size
    result << code.call(items[i])
    i += 1
  end
  result
end
```

Put in Array Class

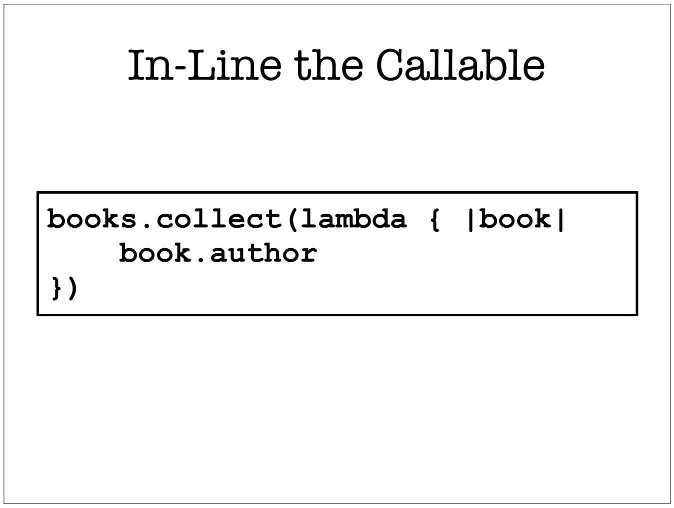
```
class Array
  def collect(code)
    result = []
    i = 0
    while i < self.size
      result << code.call(self[i])
      i += 1
    end
    result
  end
end
```

Put in Array Class

```
class GetBookAuthor
  def call(book)
    book.author
  end
end

books.collect(GetBookAuthor.new)
```

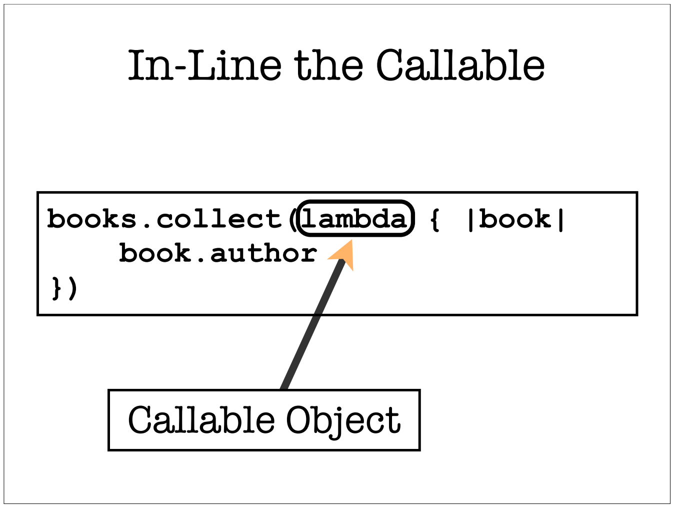
In-Line the Callable

```
books.collect(new Callable() {
  def call(book)
    book.author
  end
})
```



In-Line the Callable

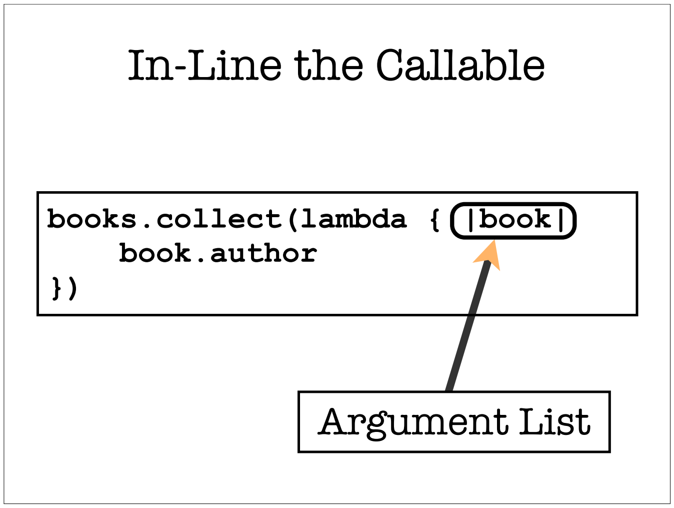
```
books.collect(lambda { |book|
    book.author
})
```



In-Line the Callable

```
books.collect(lambda { |book|
    book.author
})
```

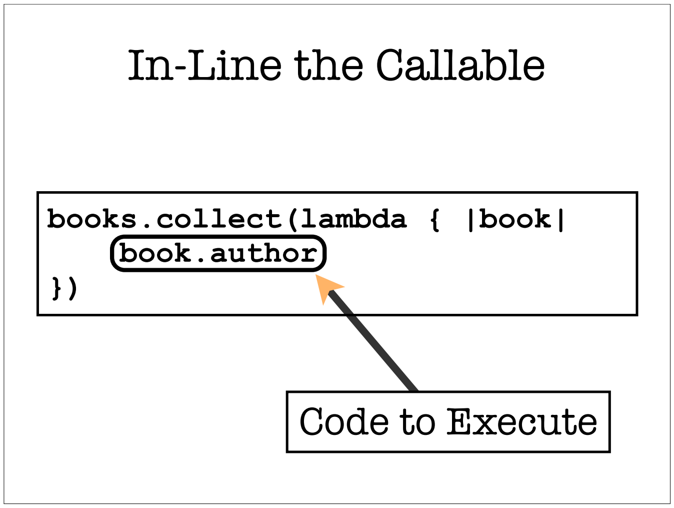
Callable Object



In-Line the Callable

```
books.collect(lambda { |book|
    book.author
})
```

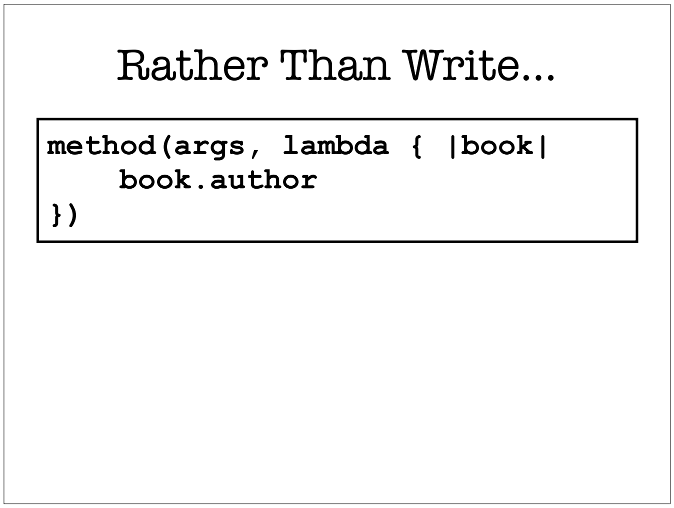
Argument List



In-Line the Callable

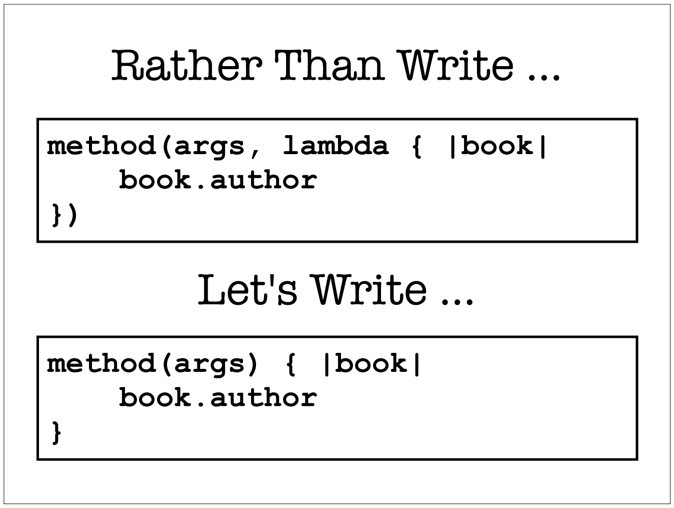
```
books.collect(lambda { |book|
    book.author
})
```

Code to Execute



Rather Than Write...

```
method(args, lambda { |book|
    book.author
})
```



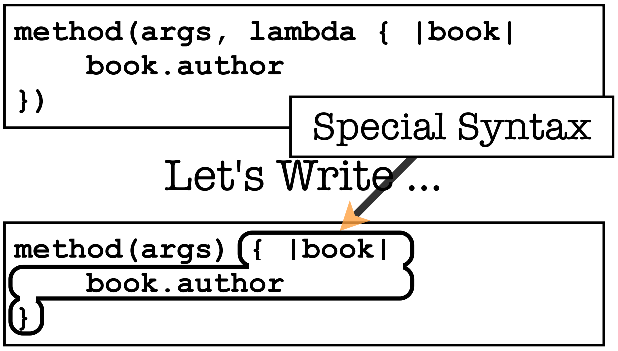
Rather Than Write ...

```
method(args, lambda { |book|
    book.author
})
```

Let's Write ...

```
method(args) { |book|
    book.author
}
```

Rather Than Write ...



```
method(args, lambda { |book|
    book.author
})
```

Special Syntax

Let's Write ...

```
method(args) { |book|
    book.author
}
```

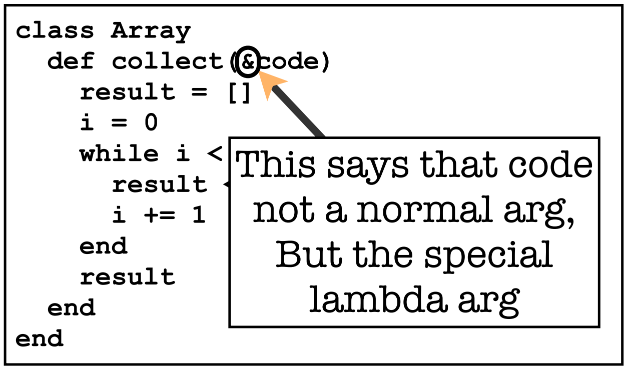
Now, Instead of This

```
class Array
  def collect(code)
    result = []
    i = 0
    while i < self.size
      result << code.call(self[i])
      i += 1
    end
    result
  end
end
```

We Write This ...

```
class Array
  def collect(&code)
    result = []
    i = 0
    while i < self.size
      result << code.call(self[i])
      i += 1
    end
    result
  end
end
```

We Write This ...



```
class Array
  def collect(&code)
    result = []
    i = 0
    while i <
      result <
      i += 1
    end
    result
  end
end
```

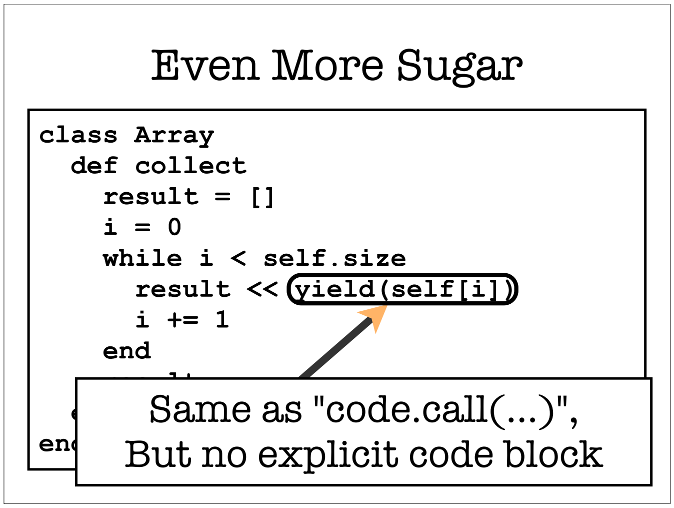
This says that code not a normal arg, But the special lambda arg

Even More Sugar

```
class Array
  def collect(&code)
    result = []
    i = 0
    while i < self.size
      result << code.call(self[i])
      i += 1
    end
    result
  end
end
```

Even More Sugar

```
class Array
  def collect
    result = []
    i = 0
    while i < self.size
      result << yield(self[i])
      i += 1
    end
    result
  end
end
```



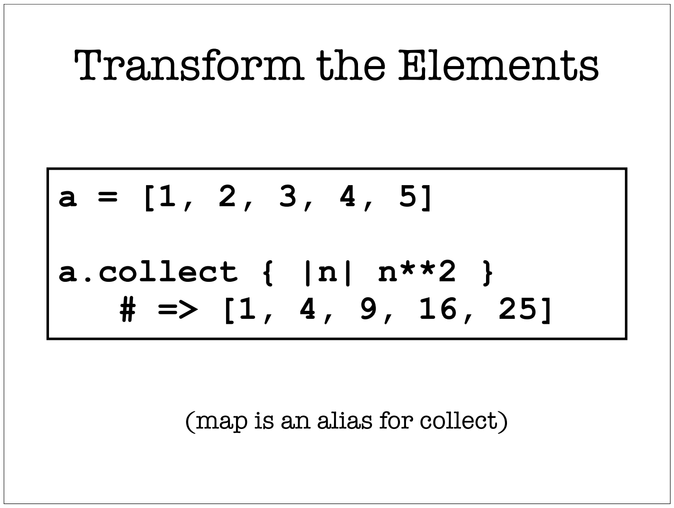
Even More Sugar

```
class Array
  def collect
    result = []
    i = 0
    while i < self.size
      result << yield(self[i])
      i += 1
    end
```

```
en
```

Same as "code.call(...)",
But no explicit code block

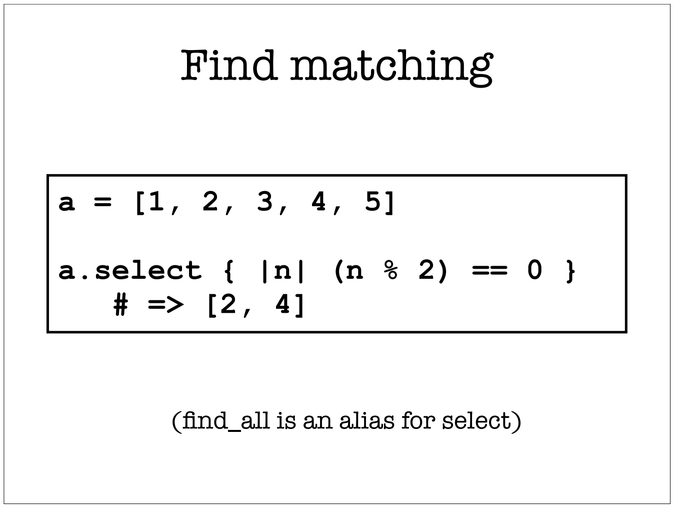
Enumerable Operations



Transform the Elements

```
a = [1, 2, 3, 4, 5]

a.collect { |n| n**2 }
   # => [1, 4, 9, 16, 25]
```

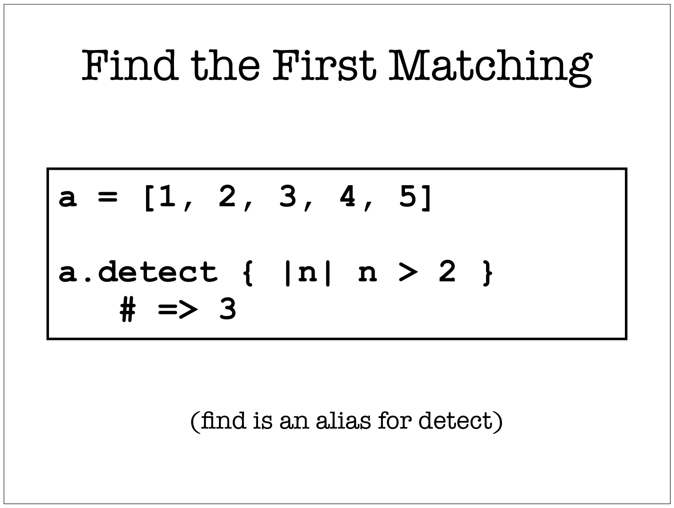
(map is an alias for collect)



Find matching

```
a = [1, 2, 3, 4, 5]

a.select { |n| (n % 2) == 0 }
   # => [2, 4]
```

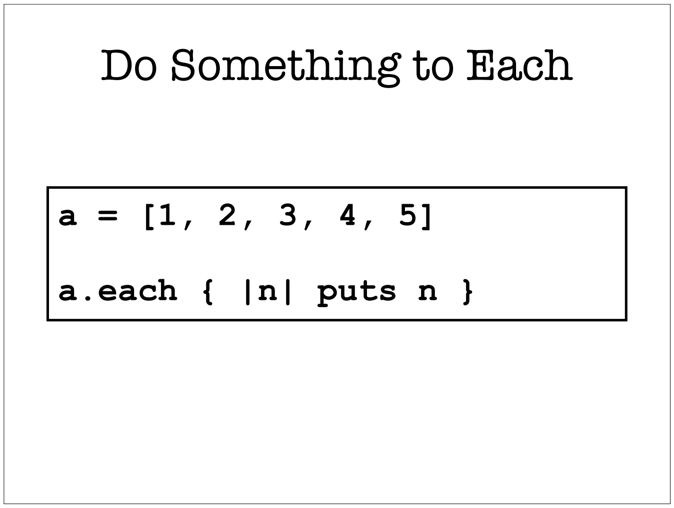
(find_all is an alias for select)



Find the First Matching

```
a = [1, 2, 3, 4, 5]

a.detect { |n| n > 2 }
   # => 3
```

(find is an alias for detect)



Do Something to Each

```
a = [1, 2, 3, 4, 5]

a.each { |n| puts n }
```

Do Something to Each

```
a = [1, 2, 3, 4, 5]

a.each_with_index { |n, i|
  puts "#{i}: #{n}"
}
```

Test the elements

```
a = [1, 2, 3, 4, 5]

a.all? { |n| n < 10 }
   # => true

a.any? { |n| (n%2) == 0 }
   # => true
```

Combine the Elements

```
a = [1, 2, 3, 4, 5]

a.inject { |accumulator, n|
  accumulator * n
}
   # => 120 (i.e. 5!)
```

Other Uses of Code Blocks

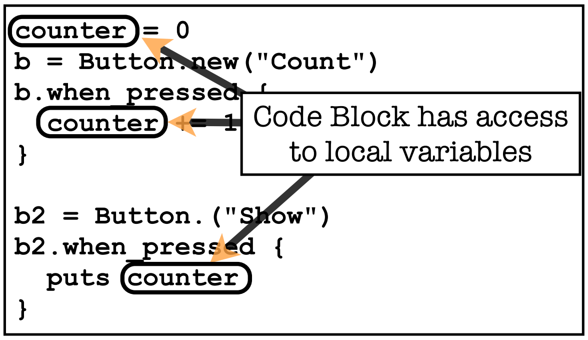
Call Backs

```
b = Button.new("Quit")
b.when_pressed { exit(0) }
```

Call Backs

```
counter = 0
b = Button.new("Count")
b.when_pressed {
  counter += 1
}

b2 = Button.("Show")
b2.when_pressed {
  puts counter
}
```

Call Backs



```
counter = 0
b = Button.new("Count")
b.when_pressed {
  counter += 1
}

b2 = Button.("Show")
b2.when_pressed {
  puts counter
}
```

Code Block has access to local variables

Try this in IRB ...

```
def make_counter
  n = 0
  lambda { n += 1 }
end

c = make_counter
c2 = make_counter
c.call    # What are the
c.call    # ... values returned
c.call    # ... for these 3 calls?

c2.call  # What's returned here?
```

Useful??

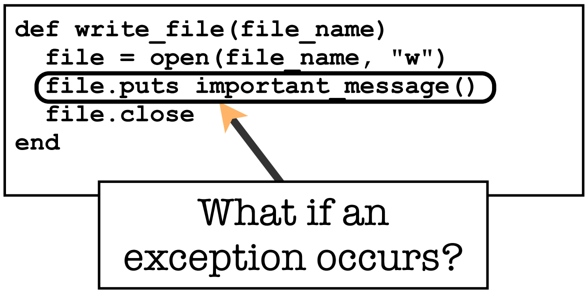
```
def make_greeter(who)
  lambda { "Hello, #{who}" }
end

g1 = make_greeter("Jim")
g2 = make_greeter("Joe")

g1.call    # => "Hello, Jim"
g2.call    # => "Hello, Joe"
```

Sandwich Code

What's Wrong With This?

```
def write_file(file_name)
  file = open(file_name, "w")
  file.puts important_message()
  file.close
end
```

What's Wrong With This?



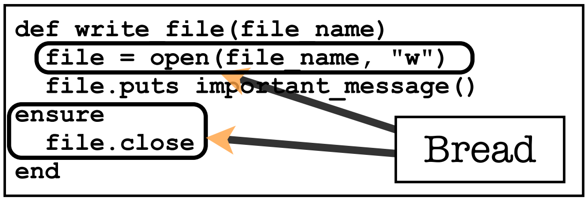
```
def write_file(file_name)
  file = open(file_name, "w")
  file.puts important_message()
  file.close
end
```

What if an exception occurs?

Better

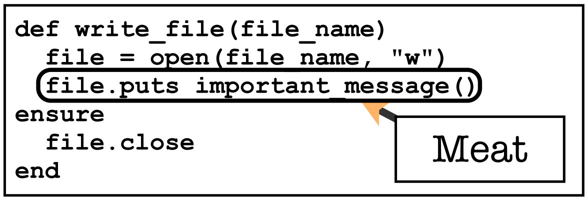
```
def write_file(file_name)
  file = open(file_name, "w")
  file.puts important_message()
ensure
  file.close
end
```

Sandwich Code



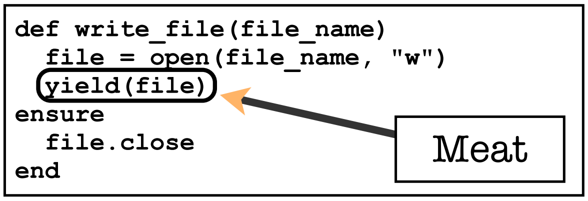
```
def write_file(file_name)
  file = open(file_name, "w")
  file.puts important_message()
ensure
  file.close
end
```

Sandwich Code



```
def write_file(file_name)
  file = open(file_name, "w")
  file.puts important_message()
ensure
  file.close
end
```

Sandwich Code



```
def write_file(file_name)
  file = open(file_name, "w")
  yield(file)
ensure
  file.close
end
```

Nice!

```
write_file("some_file.txt") { |file|
  file.puts important_message()
}
```

BTW
That's How Open Works

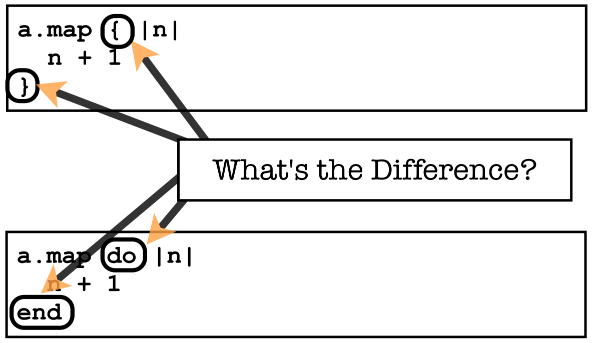
```
open("some_file.txt", "w") { |file|
  file.puts important_message()
}
```

Block Misc.

```
a.map { |n|
  n + 1
}
```

vs

```
a.map do |n|
  n + 1
end
```



```
a.map { |n|
  n + 1
}
```

What's the Difference?

```
a.map do |n|
  n + 1
end
```

```
method arg { |n| code }
```

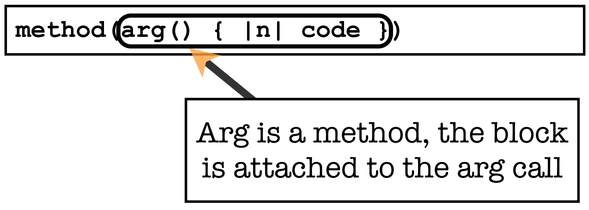
vs

```
method arg do |n| code end
```

```
method(arg() { |n| code })
```

vs

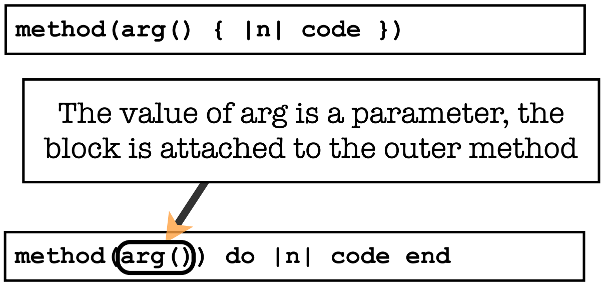
```
method(arg) do |n| code end
```



```
method(arg() { |n| code })
```

Arg is a method, the block is attached to the arg call

```
method(arg) do |n| code end
```



```
method(arg() { |n| code })
```

The value of arg is a parameter, the block is attached to the outer method

```
method(arg()) do |n| code end
```

Text Processing

File IO

Opening Files

```
open(file_name, "r") do |file|
  # read from file
end
```

```
open(file_name, "w") do |file|
  # write to file
end
```

Common Reading Idioms

```
while line = file.gets
  process_a_line(line)
end
```

Common Reading Idioms

```
all_lines = file.readlines
```

Common Reading Idioms

```
file_string = file.read
```

... or ...

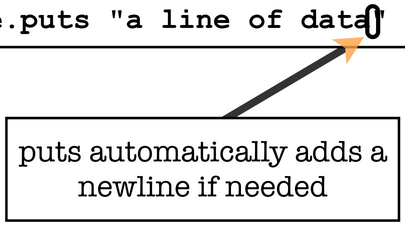
```
file_string = file.read(nbytes)
```

Common Writing Idioms

```
file.puts "a line of data"
```

Common Writing Idioms

```
file.puts "a line of data"
```



puts automatically adds a newline if needed

Common Writing Idioms

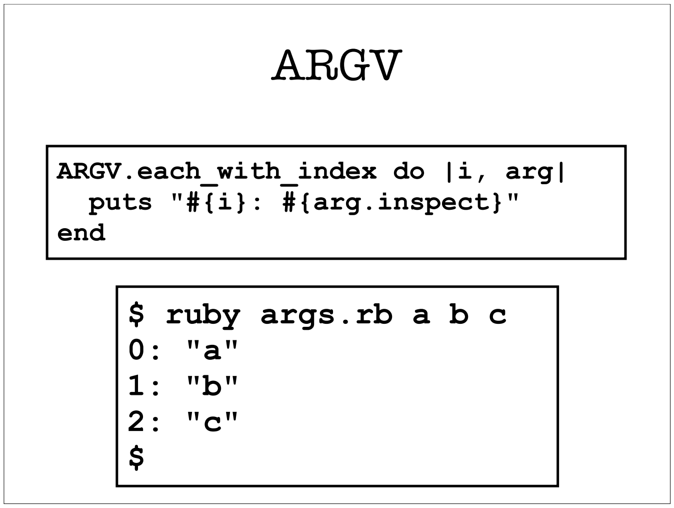
```
file.print "a line of data\n"
```

Common Writing Idioms

```
file.printf "%03d: %s\n", i, str
```

```
001: a line of data
```

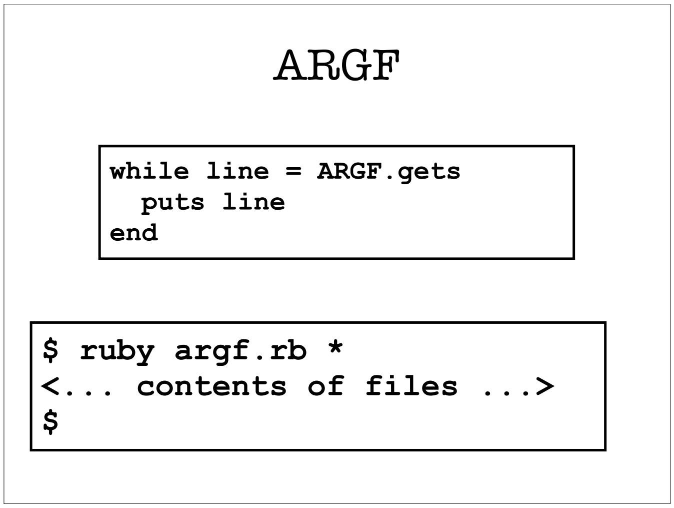
Command Line Arguments



ARGV

```
ARGV.each_with_index do |i, arg|
  puts "#{i}: #{arg.inspect}"
end
```

```
$ ruby args.rb a b c
0: "a"
1: "b"
2: "c"
$
```



ARGF

```
while line = ARGF.gets
  puts line
end
```

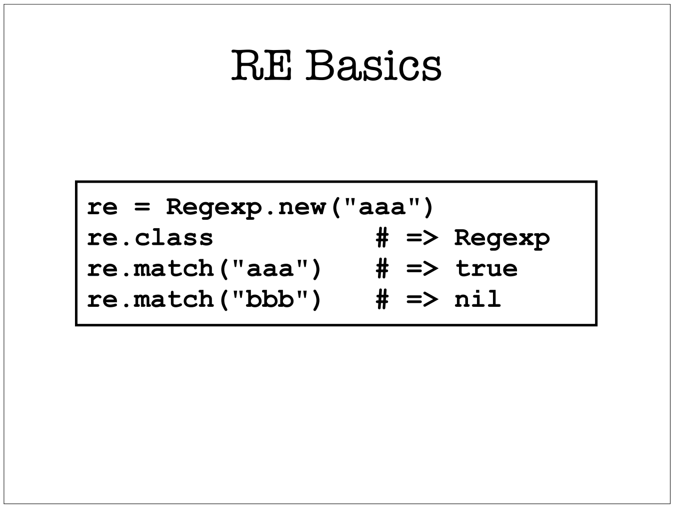
```
$ ruby argf.rb *
<... contents of files ...>
$
```

More on Command Line

OptionParser

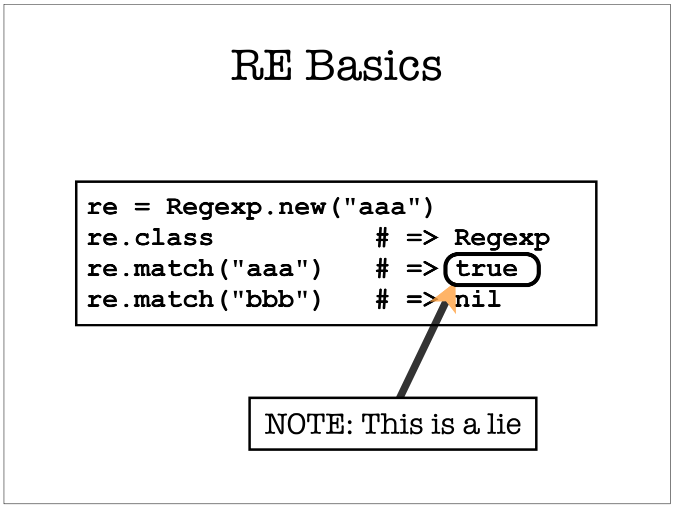
http://ruby-doc.org/stdlib/libdoc/optparse/rdoc/classes/OptionParser.html

Regular Expressions



RE Basics

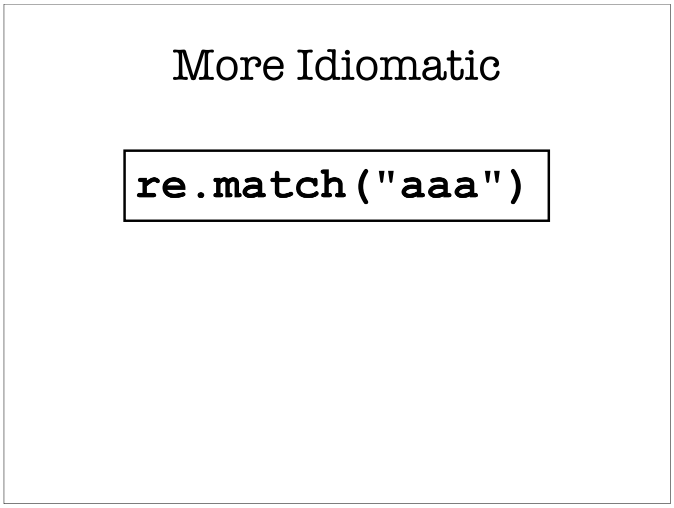
```
re = Regexp.new("aaa")
re.class          # => Regexp
re.match("aaa")   # => true
re.match("bbb")   # => nil
```



RE Basics

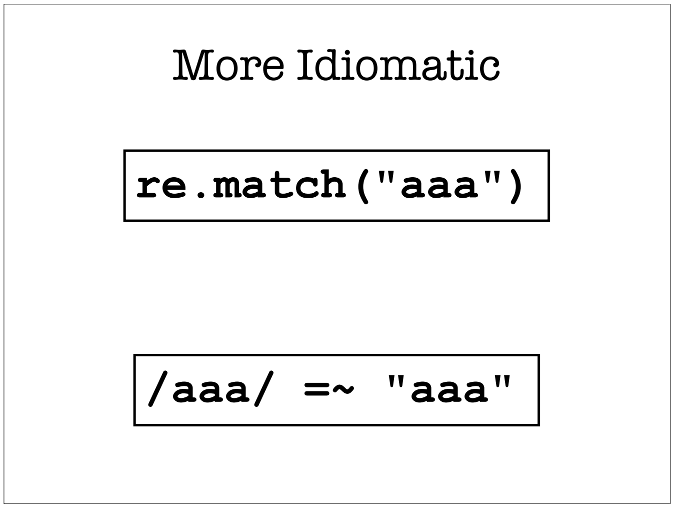
```
re = Regexp.new("aaa")
re.class          # => Regexp
re.match("aaa")   # => true
re.match("bbb")   # => nil
```

NOTE: This is a lie



More Idiomatic

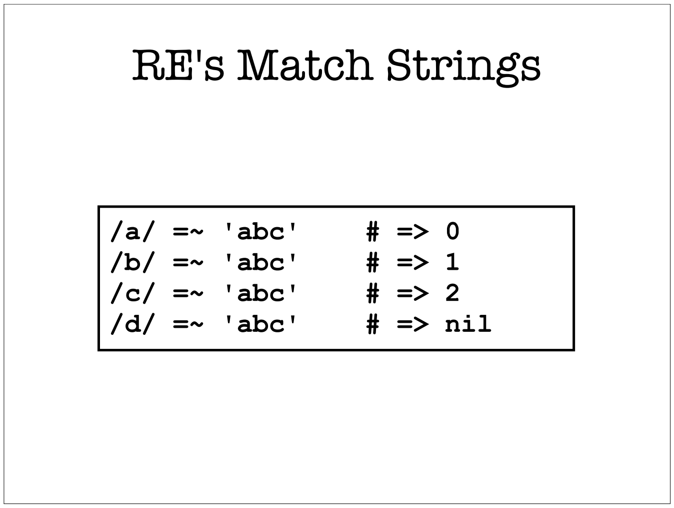
```
re.match("aaa")
```



More Idiomatic

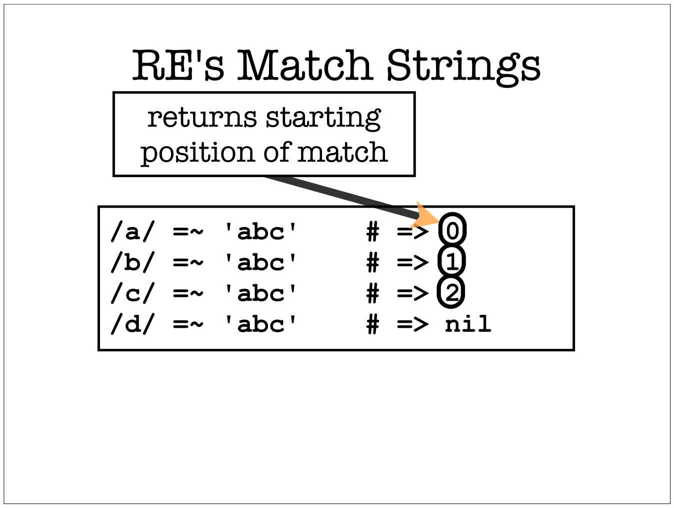
```
re.match("aaa")
```

```
/aaa/ =~ "aaa"
```



RE's Match Strings

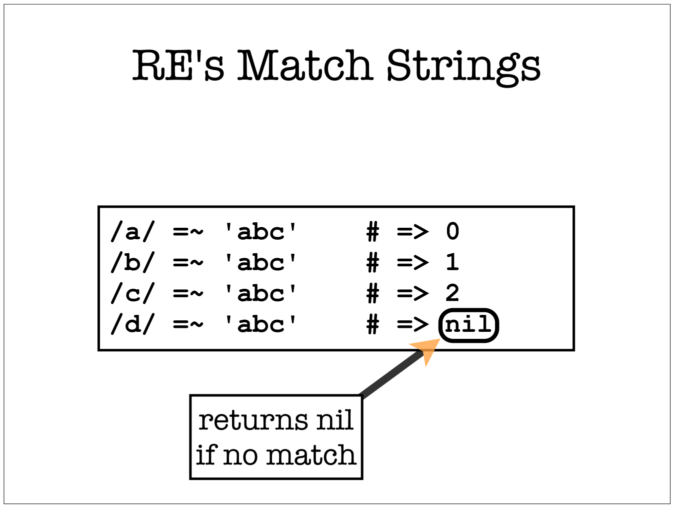
```
/a/ =~ 'abc'     # => 0
/b/ =~ 'abc'     # => 1
/c/ =~ 'abc'     # => 2
/d/ =~ 'abc'     # => nil
```



RE's Match Strings

returns starting position of match

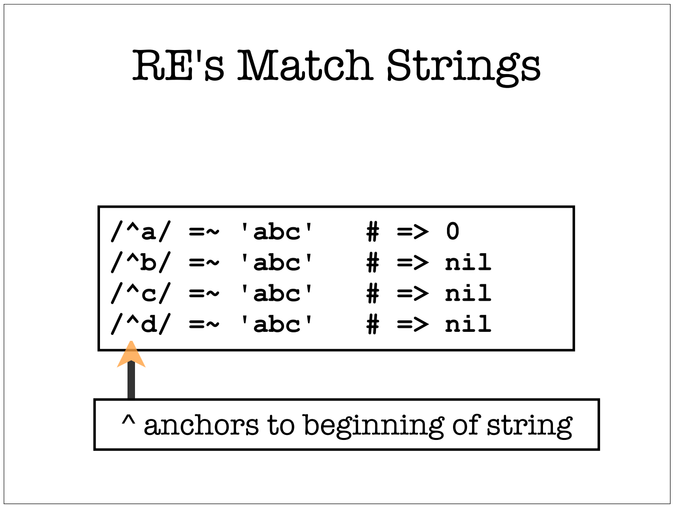
```
/a/ =~ 'abc'     # => 0
/b/ =~ 'abc'     # => 1
/c/ =~ 'abc'     # => 2
/d/ =~ 'abc'     # => nil
```



RE's Match Strings

```
/a/ =~ 'abc'     # => 0
/b/ =~ 'abc'     # => 1
/c/ =~ 'abc'     # => 2
/d/ =~ 'abc'     # => nil
```

returns nil if no match



RE's Match Strings

```
/^a/ =~ 'abc'    # => 0
/^b/ =~ 'abc'    # => nil
/^c/ =~ 'abc'    # => nil
/^d/ =~ 'abc'    # => nil
```

^ anchors to beginning of string

RE's Match Strings

```
/a$/ =~ 'abc'   # => nil
/b$/ =~ 'abc'   # => nil
/c$/ =~ 'abc'   # => 2
/d$/ =~ 'abc'   # => nil
```

$ anchors to end of string

RE's Match Strings

```
/^a*$/ =~ 'aaa'   # => 0
/^a*/  =~ 'bbb'   # => 0
/^aa*/ =~ 'bbb'   # => nil
/^a+/  =~ 'bbb'   # => nil
```

* means zero or more
+ means one or more

RE's Match Strings

```
/^a.*e$/ =~ 'apple' # => 0
/^a.*e$/ =~ 'awe'   # => 0
/^a.*e$/ =~ 'axle'  # => 0
/^a.*e$/ =~ 'all'   # => nil
```

. matches any character

RE's Match Strings

```
/^a(p|l)*e$/ =~ 'apple' # => 0
/^a(p|l)*e$/ =~ 'awe'   # => nil
/^a(p|l)*e$/ =~ 'axle'  # => nil
/^a(p|l)*e$/ =~ 'all'   # => nil
```

( ) provides grouping
| separates alternatives

RE's Match Strings

```
/^a[pl]*e$/ =~ 'apple' # => 0
/^a[pl]*e$/ =~ 'awe'   # => nil
/^a[pl]*e$/ =~ 'axle'  # => nil
/^a[pl]*e$/ =~ 'all'   # => nil
```

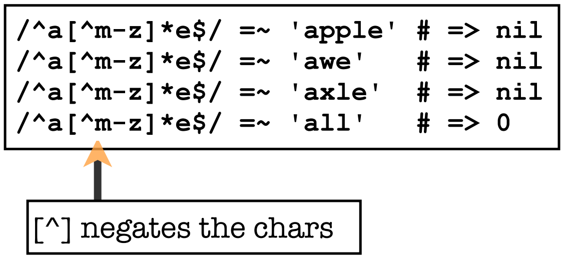
[...] matches any char in list

RE's Match Strings

```
/^a[m-z]*e$/ =~ 'apple' # => nil
/^a[m-z]*e$/ =~ 'awe'   # => 0
/^a[m-z]*e$/ =~ 'axle'  # => nil
/^a[m-z]*e$/ =~ 'all'   # => nil
```

[-] is a range of chars

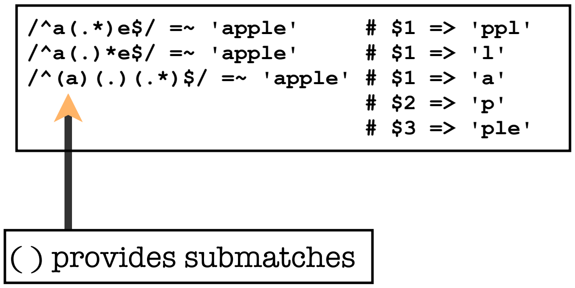
RE's Match Strings



```
/^a[^m-z]*e$/ =~ 'apple' # => nil
/^a[^m-z]*e$/ =~ 'awe'   # => nil
/^a[^m-z]*e$/ =~ 'axle'  # => nil
/^a[^m-z]*e$/ =~ 'all'   # => 0
```

[^] negates the chars

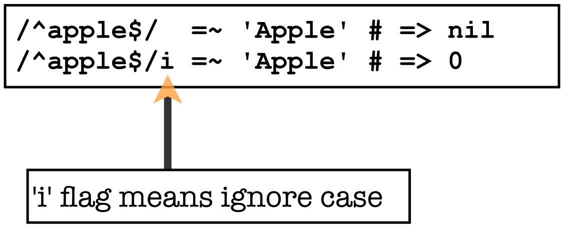
RE's Match Strings



```
/^a(.*)e$/ =~ 'apple'     # $1 => 'ppl'
/^a(.)*e$/ =~ 'apple'     # $1 => 'l'
/^(a)(.)(.*)$/ =~ 'apple' # $1 => 'a'
                          # $2 => 'p'
                          # $3 => 'ple'
```

( ) provides submatches

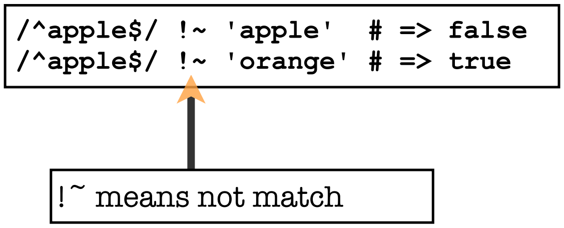
RE's Match Strings



```
/^apple$/  =~ 'Apple' # => nil
/^apple$/i =~ 'Apple' # => 0
```

'i' flag means ignore case

RE's Match Strings



```
/^apple$/ !~ 'apple'  # => false
/^apple$/ !~ 'orange' # => true
```

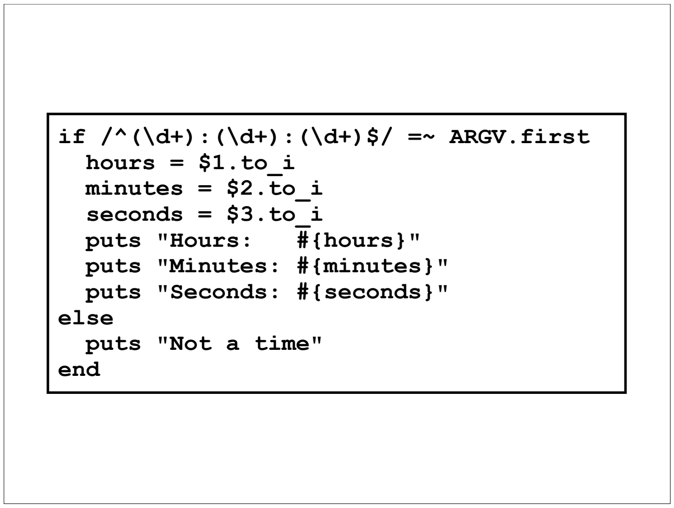
!˜ means not match

Other Regex Stuff

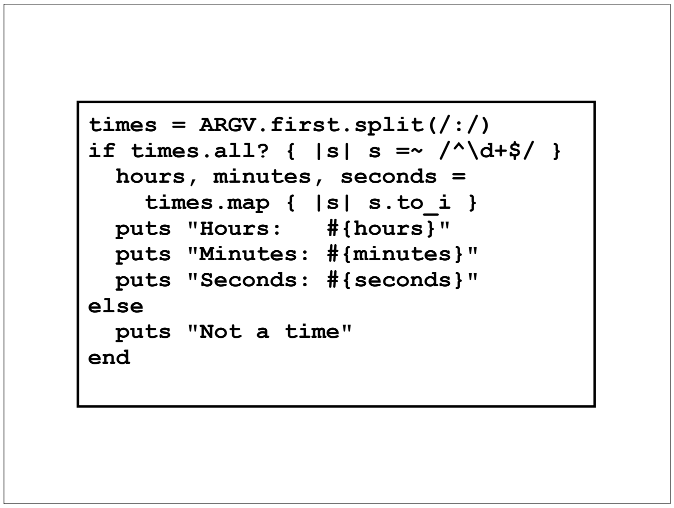
- Use {n}, {n,}, {n,m} to specify number of repetitions
- Use (?: ...) to turn off captures
- Escape special chars with \
- Special Patterns
  - \s, \S, \w, \W, \d, \D, \A, \Z

http://rubular.com/

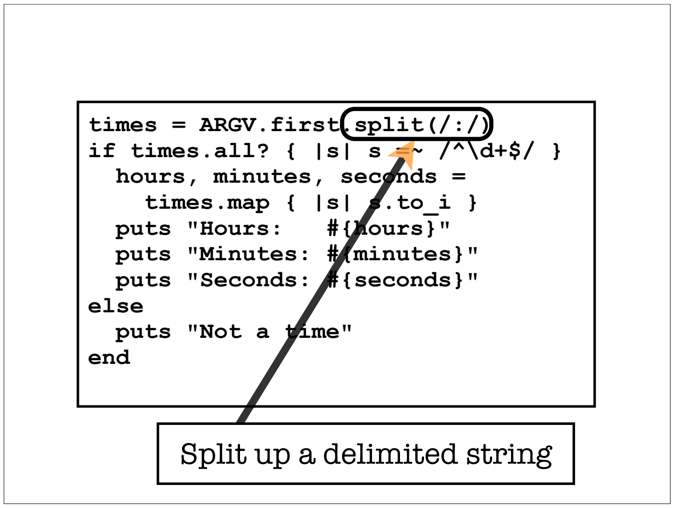
Using RegExp



```
if /^(\d+):(\d+):(\d+)$/ =~ ARGV.first
  hours = $1.to_i
  minutes = $2.to_i
  seconds = $3.to_i
  puts "Hours:   #{hours}"
  puts "Minutes: #{minutes}"
  puts "Seconds: #{seconds}"
else
  puts "Not a time"
end
```

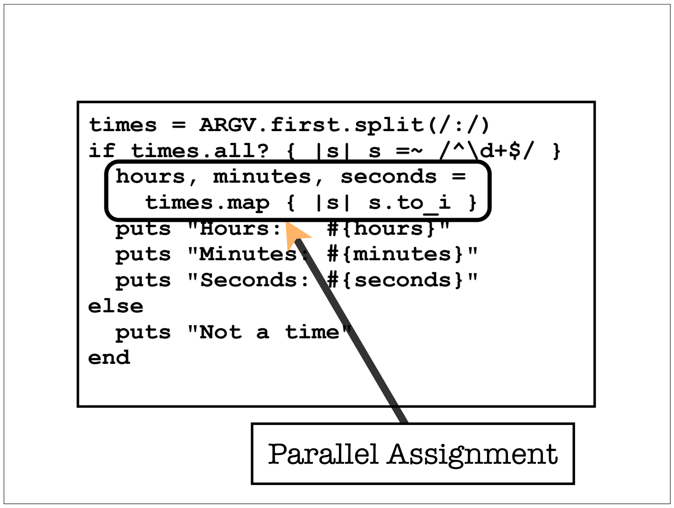


```
times = ARGV.first.split(/:/)
if times.all? { |s| s =~ /^\d+$/ }
  hours, minutes, seconds =
    times.map { |s| s.to_i }
  puts "Hours:   #{hours}"
  puts "Minutes: #{minutes}"
  puts "Seconds: #{seconds}"
else
  puts "Not a time"
end
```



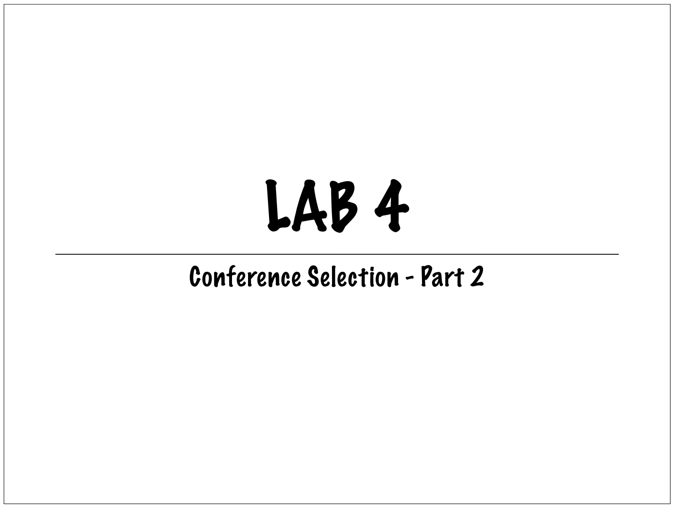
```
times = ARGV.first.split(/:/)
if times.all? { |s| s =~ /^\d+$/ }
  hours, minutes, seconds =
    times.map { |s| s.to_i }
  puts "Hours:   #{hours}"
  puts "Minutes: #{minutes}"
  puts "Seconds: #{seconds}"
else
  puts "Not a time"
end
```

Split up a delimited string



```
times = ARGV.first.split(/:/)
if times.all? { |s| s =~ /^\d+$/ }
  hours, minutes, seconds =
    times.map { |s| s.to_i }
  puts "Hours:   #{hours}"
  puts "Minutes: #{minutes}"
  puts "Seconds: #{seconds}"
else
  puts "Not a time"
end
```

Parallel Assignment



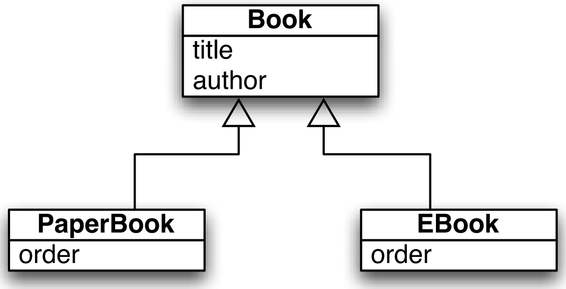
LAB 4

Conference Selection - Part 2

Inheritance

Book Store

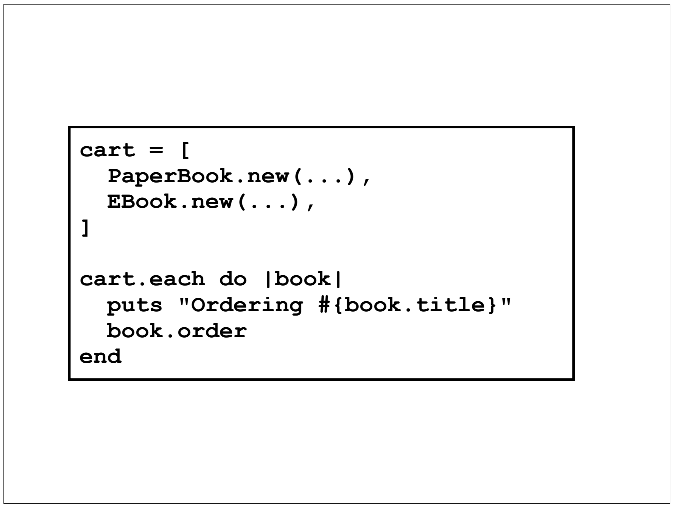
- Two kinds of books
  - Paper Books
    - Ordered by sending a request to the fulfillment organization
  - E-Books
    - Ordered by initiating a download



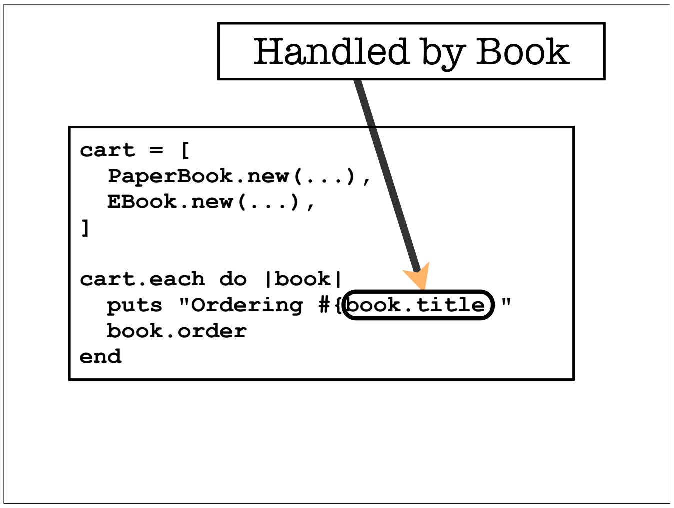
```
class Book
  attr_reader :title, :author, :isbn
end
```

```
class PaperBook < Book
  def order
    send_fulfillment_request(isbn)
  end
end
```

```
class EBook < Book
  def order
    initiate_download(isbn)
  end
end
```

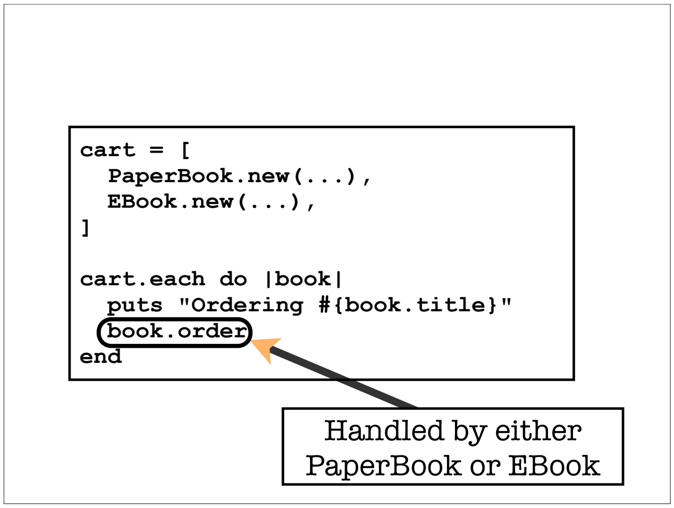


```
cart = [
  PaperBook.new(...),
  EBook.new(...),
]

cart.each do |book|
  puts "Ordering #{book.title}"
  book.order
end
```



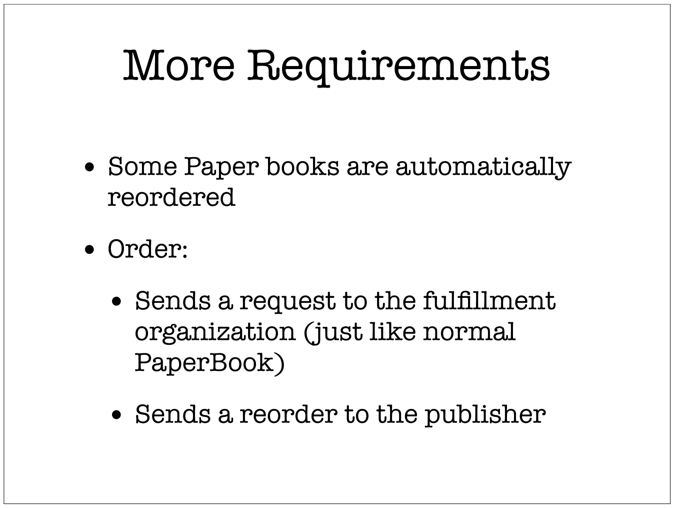
Handled by Book

```
cart = [
  PaperBook.new(...),
  EBook.new(...),
]

cart.each do |book|
  puts "Ordering #{book.title}"
  book.order
end
```



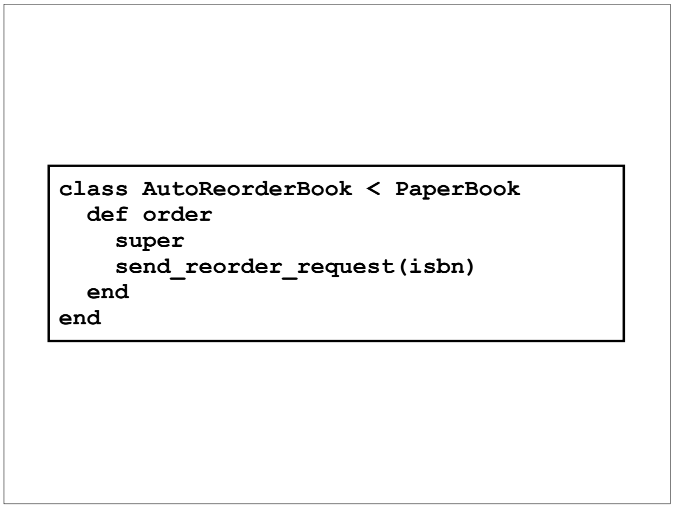
```
cart = [
  PaperBook.new(...),
  EBook.new(...),
]

cart.each do |book|
  puts "Ordering #{book.title}"
  book.order
end
```

Handled by either PaperBook or EBook

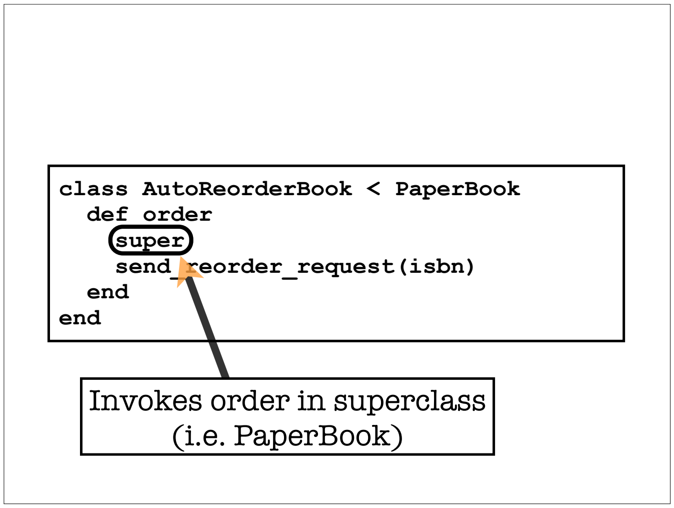


More Requirements

- Some Paper books are automatically reordered
- Order:
  - Sends a request to the fulfillment organization (just like normal PaperBook)
  - Sends a reorder to the publisher



```
class AutoReorderBook < PaperBook
  def order
    super
    send_reorder_request(isbn)
  end
end
```



```
class AutoReorderBook < PaperBook
  def order
    super
    send_reorder_request(isbn)
  end
end
```

Invokes order in superclass (i.e. PaperBook)

Some super Notes

```
def f(a, b)
  super(a, b) # same args
  super       # same as super(a,b)
  super(a)    # different args
end
```

More super Notes

- You cannot:
  - Call a different method in the super class
  - Call the method in a grandfather class
    - (i.e. can't skip parent classes)

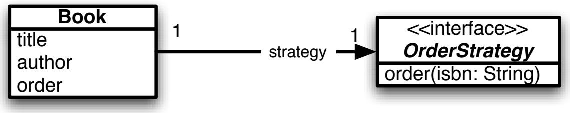
More super Notes

- Super is not a reference to an object of the parent class
  - i.e. You cannot:

```
  super.some_parent_method
```

Back to the Books

- We realize that our design is rather limiting
  - Many books are available in both paper and electronic format

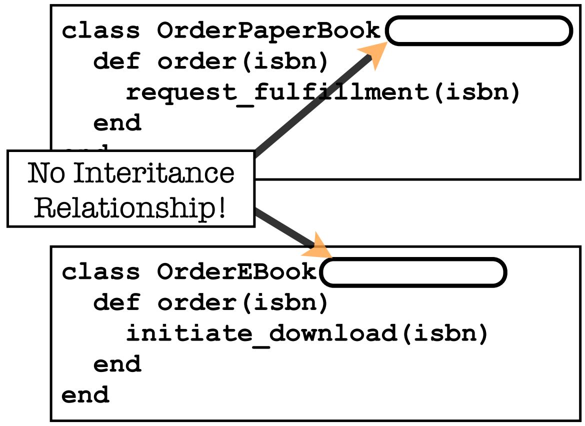


```
class Book
  attr_reader
    :title, :author, :isbn

  def order
    @strategy.order(isbn)
  end
end
```

How do we write an interface in Ruby?

```
class OrderPaperBook
  def order(isbn)
    request_fulfillment(isbn)
  end
end
```

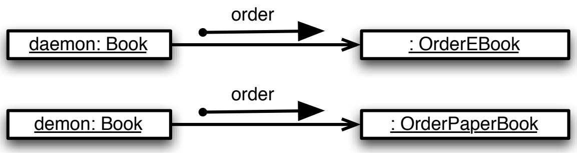
```
class OrderEBook
  def order(isbn)
    initiate_download(isbn)
  end
end
```

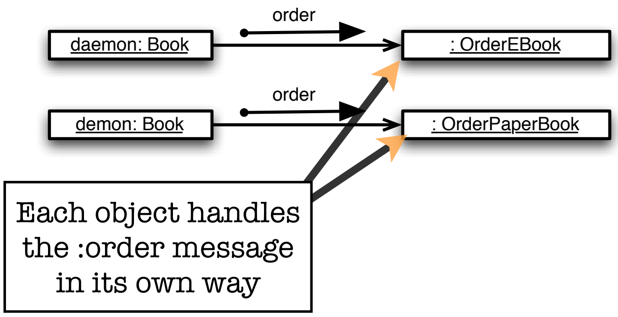


```
class OrderPaperBook
  def order(isbn)
    request_fulfillment(isbn)
  end
end
```

No Interitance Relationship!

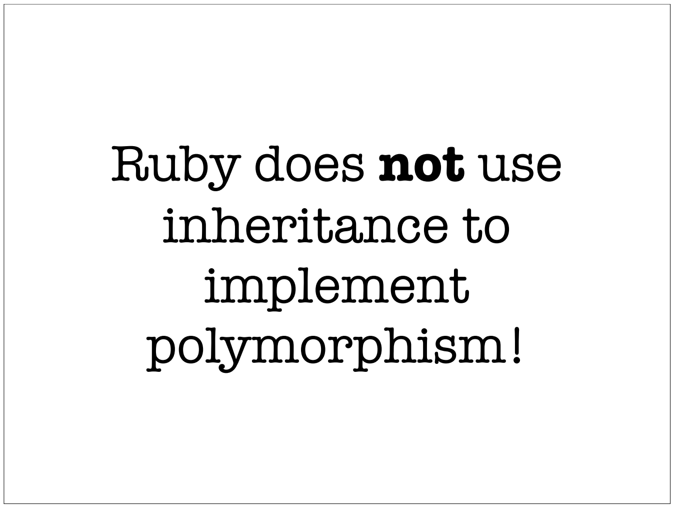
```
class OrderEBook
  def order(isbn)
    initiate_download(isbn)
  end
end
```



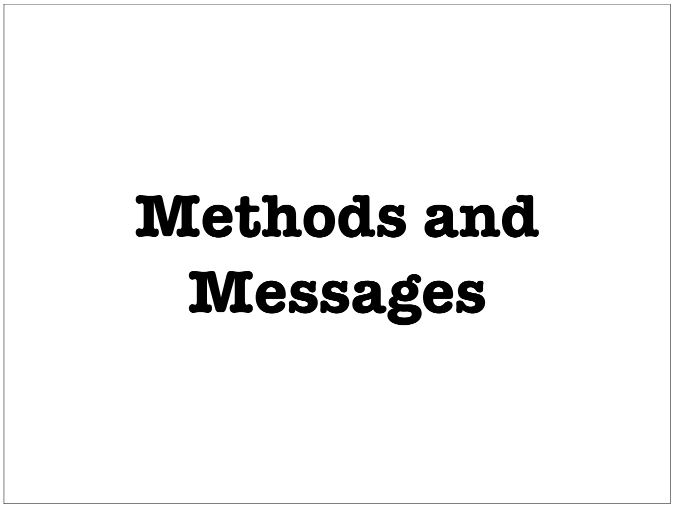


Each object handles the :order message in its own way

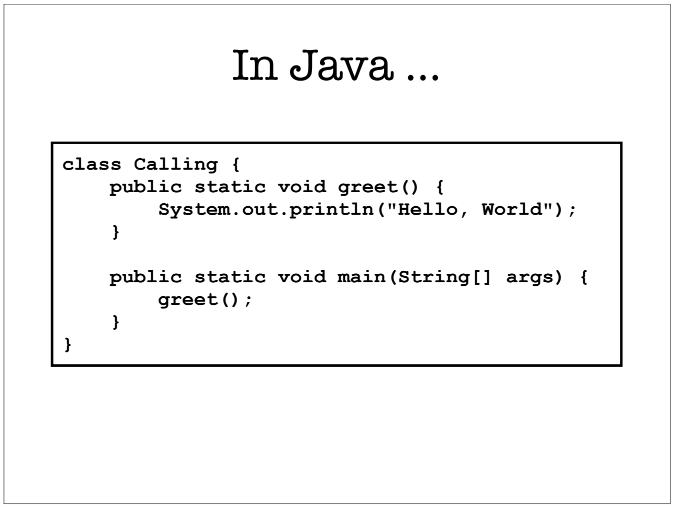
Duck Typing



Ruby does **not** use inheritance to implement polymorphism!

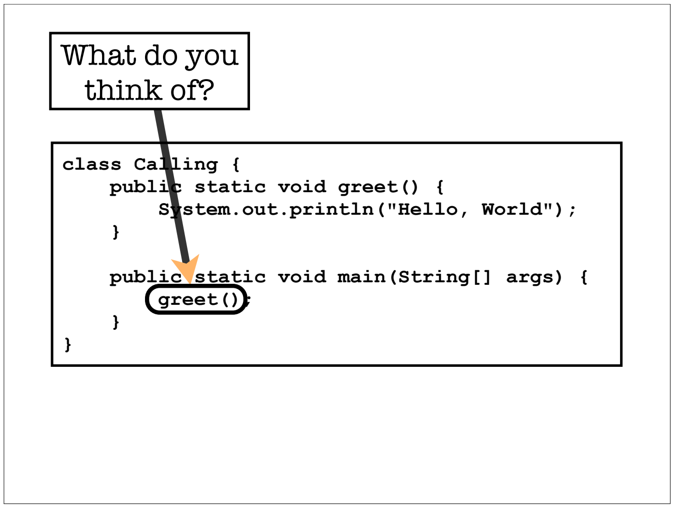


Methods and Messages



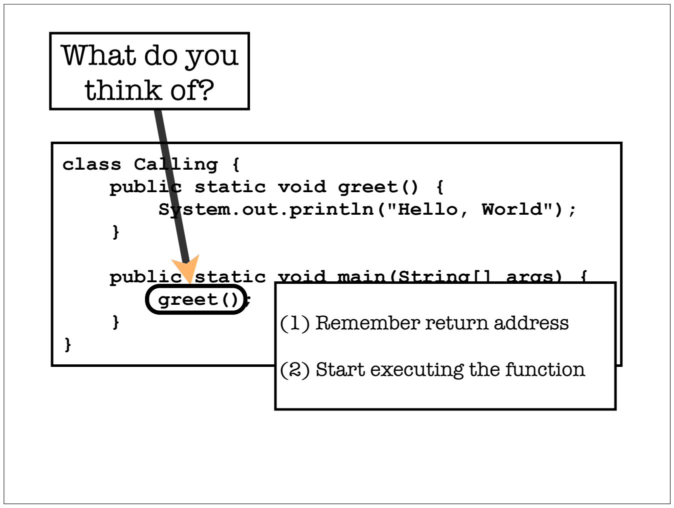
In Java ...

```
class Calling {
    public static void greet() {
        System.out.println("Hello, World");
    }

    public static void main(String[] args) {
        greet();
    }
}
```



What do you think of?

```
class Calling {
    public static void greet() {
        System.out.println("Hello, World");
    }

    public static void main(String[] args) {
        greet();
    }
}
```

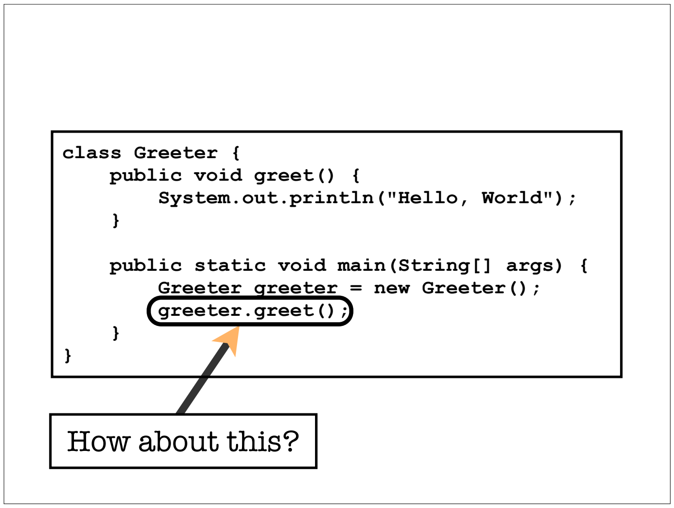


What do you think of?

```
class Calling {
    public static void greet() {
        System.out.println("Hello, World");
    }

    public static void main(String[] args) {
        greet();
    }
}
```

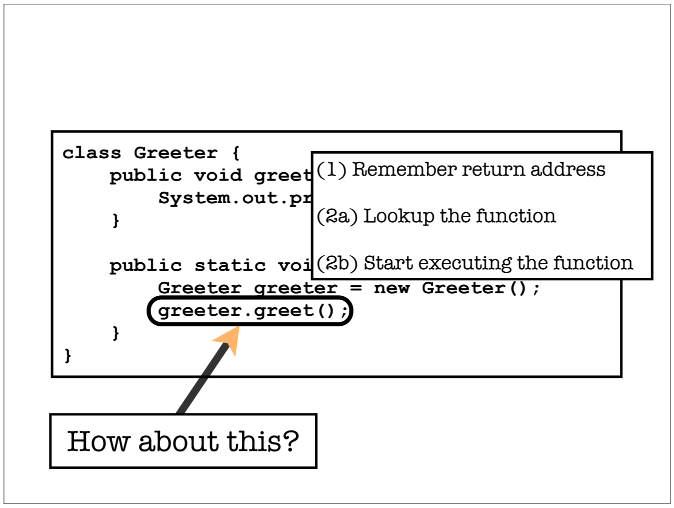
(1) Remember return address

(2) Start executing the function



```
class Greeter {
    public void greet() {
        System.out.println("Hello, World");
    }

    public static void main(String[] args) {
        Greeter greeter = new Greeter();
        greeter.greet();
    }
}
```

How about this?



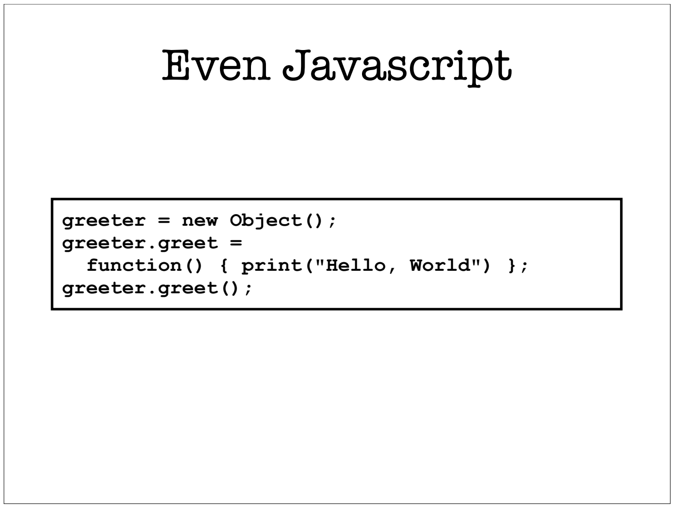
```
class Greeter {
    public void greet
        System.out.pr
    }

    public static voi
        Greeter greeter = new Greeter();
        greeter.greet();
    }
}
```

(1) Remember return address

(2a) Lookup the function

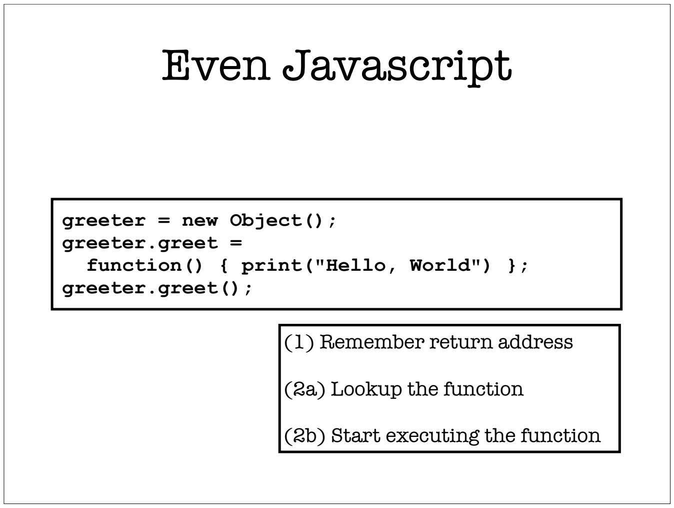
(2b) Start executing the function

How about this?



Even Javascript

```
greeter = new Object();
greeter.greet =
  function() { print("Hello, World") };
greeter.greet();
```



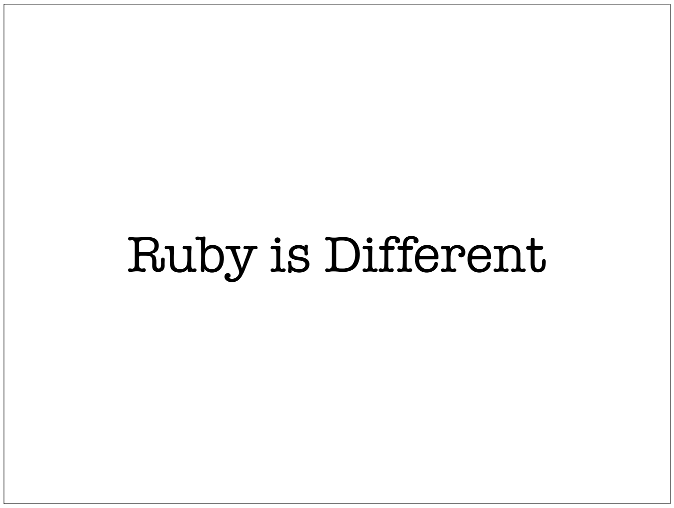
Even Javascript

```
greeter = new Object();
greeter.greet =
  function() { print("Hello, World") };
greeter.greet();
```

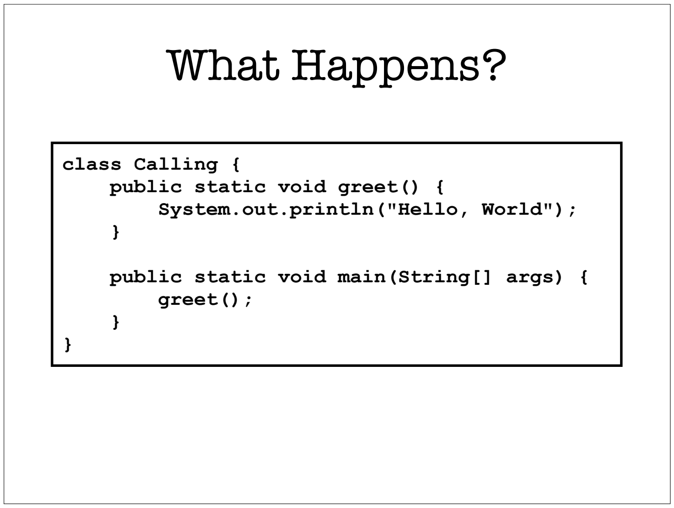
(1) Remember return address

(2a) Lookup the function

(2b) Start executing the function

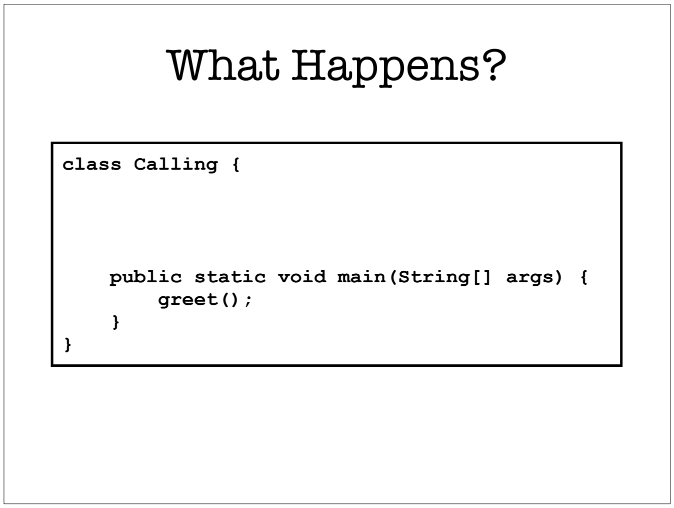


Ruby is Different



What Happens?

```
class Calling {
    public static void greet() {
        System.out.println("Hello, World");
    }

    public static void main(String[] args) {
        greet();
    }
}
```



What Happens?

```
class Calling {



    public static void main(String[] args) {
        greet();
    }
}
```

What Happens?

```
class Calling {



    public static void main(String[] args) {
        greet();
    }
}
```

```
Greeter.java:8: cannot find symbol
symbol  : method greet()
location: class Greeter
        greeter.greet();
                ^
1 error
```

What Happens?

```
greeter = new Object();
greeter.greet =
  function() { print("Hello, World") };
greeter.greet();
```

What Happens?

```
greeter = new Object();


greeter.greet();
```

What Happens?

```
greeter = new Object();


greeter.greet();
```

```
greeter.js:3: TypeError:
greeter.greet is not a function
```

What Happens?

```
class Greeter
  def greet
    puts "Hello, World"
  end
end

greeter = Greeter.new
greeter.greet
```

What Happens?

```
class Greeter



end

greeter = Greeter.new
greeter.greet
```

What Happens?

```
class Greeter



end

greeter = Greeter.new
greeter.greet
```

```
greeter.rb:8: undefined method `greet'
for #<Greeter:0x293c4> (NoMethodError)
```

What Happens?

```
class Greeter
  def method_missing(sym, *args, &block)
    puts "Sorry, I'm confused!"
  end
end

greeter = Greeter.new
greeter.greet
```

What Happens?

```
class Greeter
  def method_missing(sym, *args, &block)
    puts "Sorry, I'm confused!"
  end
end

greeter = Greeter.new
greeter.greet
```

```
Sorry, I'm confused!
```

- Send a message to an object
- Lookup a method for the message
  - If found, execute it
  - If not found, send a method_missing message

What's a Message?

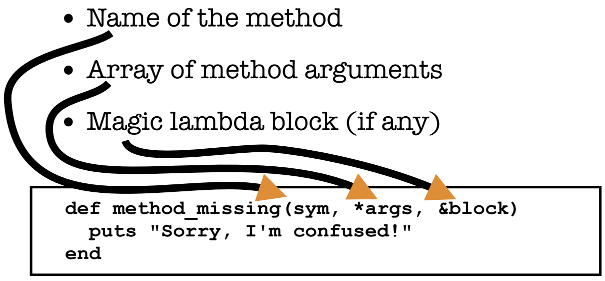
- Name of the method
- Array of method arguments
- Magic lambda block (if any)

What's a Message?

- Name of the method
- Array of method arguments
- Magic lambda block (if any)

```
  def method_missing(sym, *args, &block)
    puts "Sorry, I'm confused!"
  end
```

What's a Message?



- Name of the method
- Array of method arguments
- Magic lambda block (if any)

```
  def method_missing(sym, *args, &block)
    puts "Sorry, I'm confused!"
  end
```

Why is that useful?

```
class VCR
  def initialize
    @messages = []
  end

  def method_missing(sym, *args, &block)
    @messages << [sym, args, block]
  end

  ...
end
```

```
vcr = VCR.new
vcr.upcase!
vcr.sub!(/world/i, 'Universe')
```

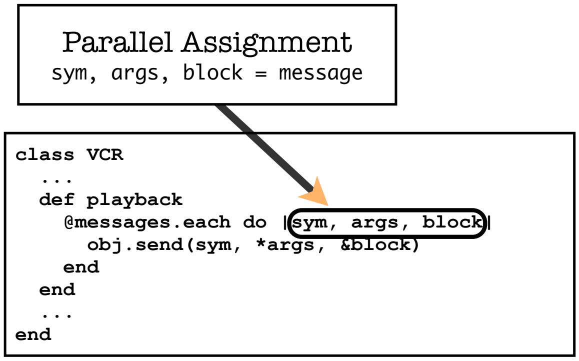
```
vcr = VCR.new
vcr.upcase!
vcr.sub!(/world/i, 'Universe')
```

@messages[0]: [:upcase!, [], nil]

```
vcr = VCR.new
vcr.upcase!
vcr.sub!(/world/i, 'Universe')
```

@messages[0]: [:upcase!, [], nil]
@messages[1]: [:sub!, [/world/i, 'Universe'], nil]

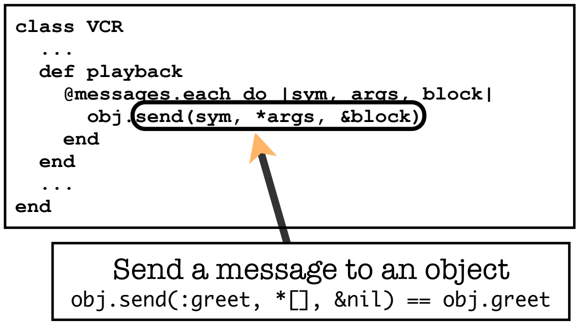
```
class VCR
  ...
  def playback(obj)
    @messages.each do |sym, args, block|
      obj.send(sym, *args, &block)
    end
  end
  ...
end
```



Parallel Assignment

```
sym, args, block = message
```

```
class VCR
  ...
  def playback
    @messages.each do |sym, args, block|
      obj.send(sym, *args, &block)
    end
  end
  ...
end
```



```
class VCR
  ...
  def playback
    @messages.each do |sym, args, block|
      obj.send(sym, *args, &block)
    end
  end
  ...
end
```

Send a message to an object

```
obj.send(:greet, *[], &nil) == obj.greet
```

```
s = "Hello, World"
vcr.playback(s)

puts s   # => ?
```

@messages[0]: [:upcase!, [], nil]
@messages[1]: [:sub!, [/world/i, 'Universe'], nil]

```
s = "Hello, World"
vcr.playback(s)

puts s   # => "HELLO, Universe"
```

@messages[0]: [:upcase!, [], nil]
@messages[1]: [:sub!, [/world/i, 'Universe'], nil]

Ideas

- Message Recorders
- Proxy Objects
- Mock Objects
- Dynamic Methods

```
class SuperHash < Hash
  def method_missing(sym, *args, &block)
    self[sym]
  end
end
```

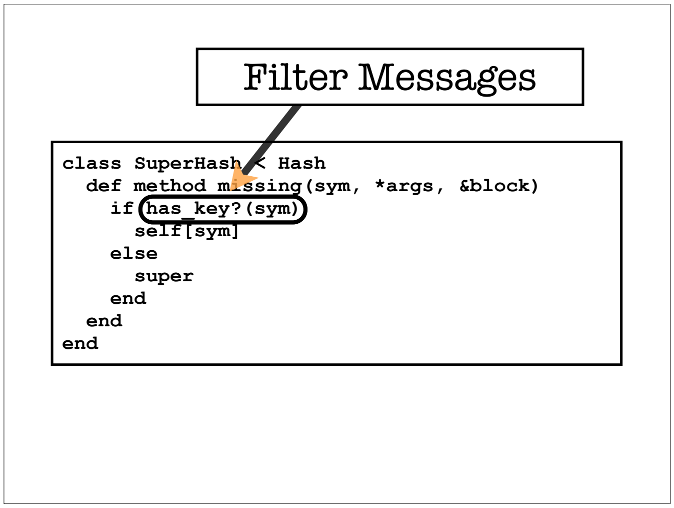
```
$ irb --simple-prompt
>> require 'super_hash'
=> true
>> h = SuperHash.new
=> {}
```

```
>> h[:stuff] = "HI"
=> "HI"
>> h[:stuff]
=> "HI"
>> h.stuff
=> "HI"
```

```
>> h.not_there
=> nil
```

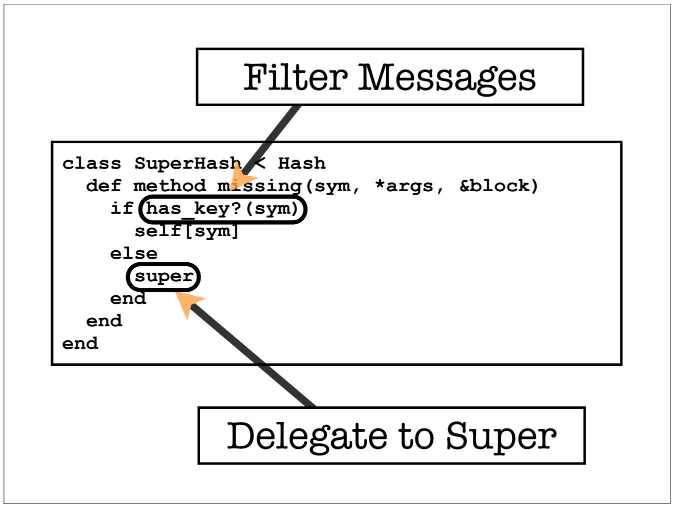
```
class SuperHash < Hash
  def method_missing(sym, *args, &block)
    self[sym]
  end
end
```

```
class SuperHash < Hash
  def method_missing(sym, *args, &block)
    if has_key?(sym)
      self[sym]
    else
      super
    end
  end
end
```



Filter Messages

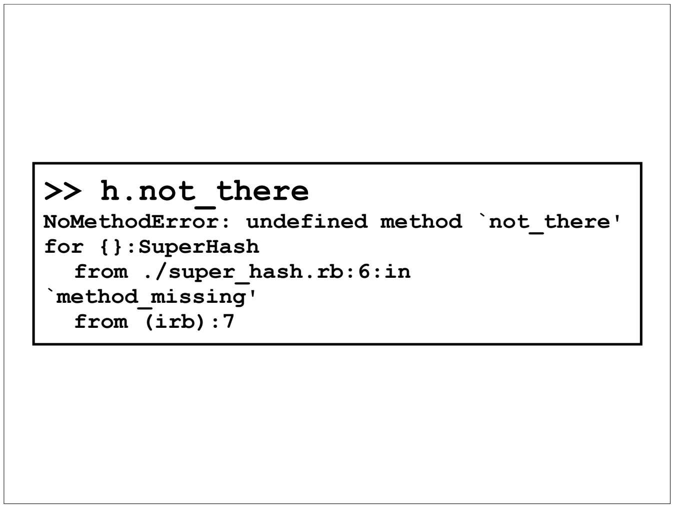
```
class SuperHash < Hash
  def method_missing(sym, *args, &block)
    if has_key?(sym)
      self[sym]
    else
      super
    end
  end
end
```



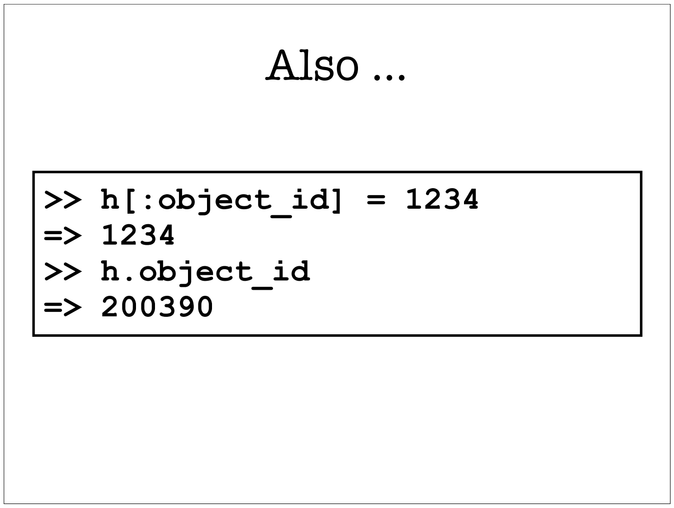
Filter Messages

```
class SuperHash < Hash
  def method_missing(sym, *args, &block)
    if has_key?(sym)
      self[sym]
    else
      super
    end
  end
end
```

Delegate to Super

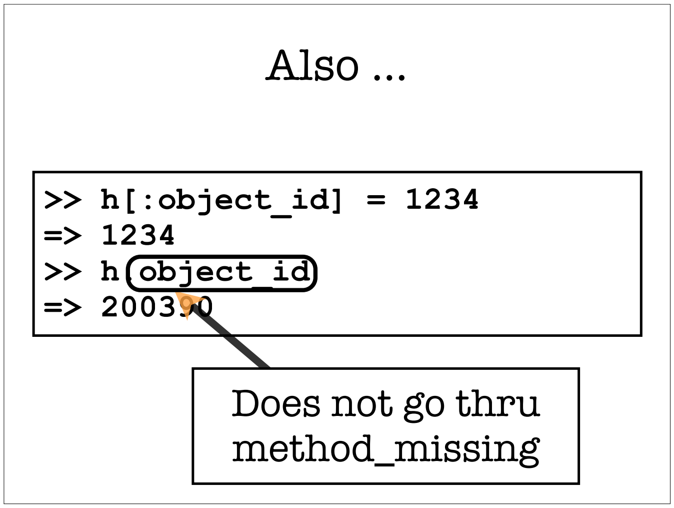


```
>> h.not_there
NoMethodError: undefined method `not_there'
for {}:SuperHash
   from ./super_hash.rb:6:in
`method_missing'
   from (irb):7
```



Also ...

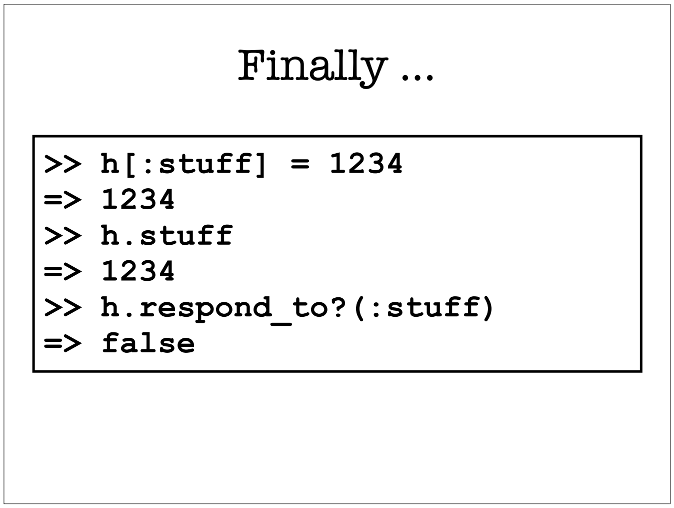
```
>> h[:object_id] = 1234
=> 1234
>> h.object_id
=> 200390
```



Also ...

```
>> h[:object_id] = 1234
=> 1234
>> h.object_id
=> 200390
```

Does not go thru
method_missing



Finally ...

```
>> h[:stuff] = 1234
=> 1234
>> h.stuff
=> 1234
>> h.respond_to?(:stuff)
=> false
```

```
class SuperHash < Hash
  ...
  def respond_to?(sym)
    has_key?(sym) || super
  end
  ...
end
```

Using Method Missing

- Filter on messages you want to handle
- Delegate un-handled messages to super
- Beware of predefined methods
  - (BlankSlate/BasicObject)
- Implement respond_to?
- Use lightly!

LAB 5

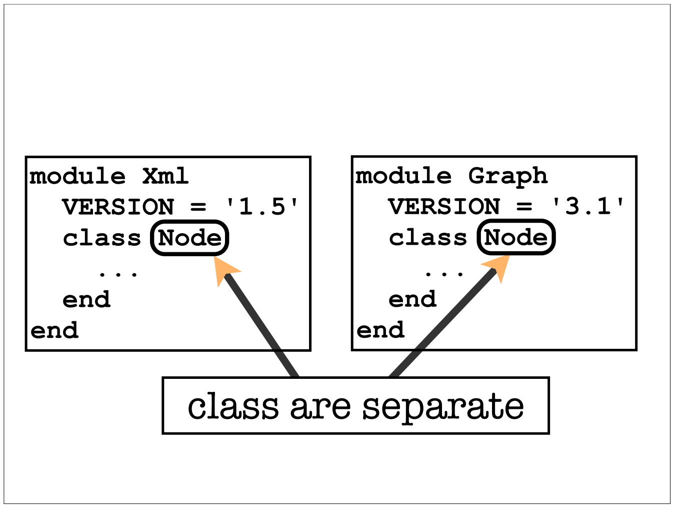
Read-Only Proxies

Modules

Name Spaces

```
module Xml
  VERSION = '1.5'
  class Node
    ...
  end
end
```

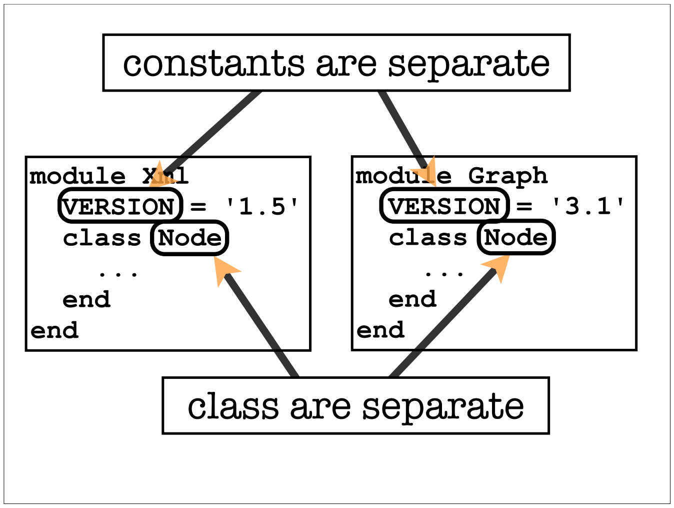
```
module Graph
  VERSION = '3.1'
  class Node
    ...
  end
end
```



```
module Xml
  VERSION = '1.5'
  class Node
    ...
  end
end
```

```
module Graph
  VERSION = '3.1'
  class Node
    ...
  end
end
```

class are separate

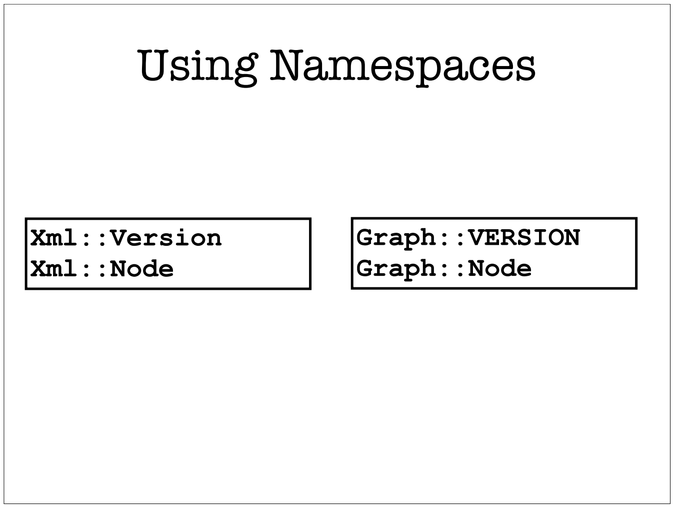


constants are separate

```
module Xml
  VERSION = '1.5'
  class Node
    ...
  end
end
```

```
module Graph
  VERSION = '3.1'
  class Node
    ...
  end
end
```

class are separate

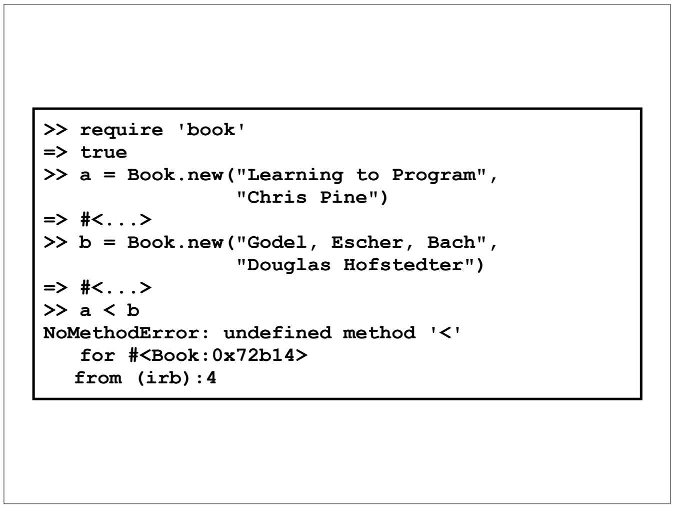


Using Namespaces

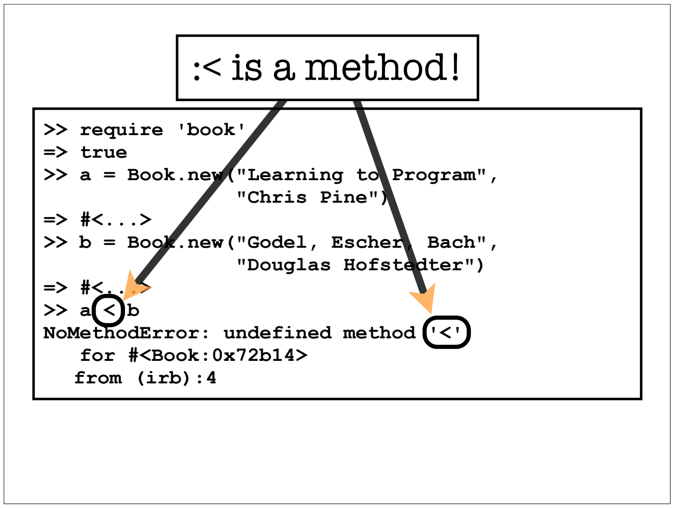
```
Xml::Version
Xml::Node
```

```
Graph::VERSION
Graph::Node
```

Mix-ins

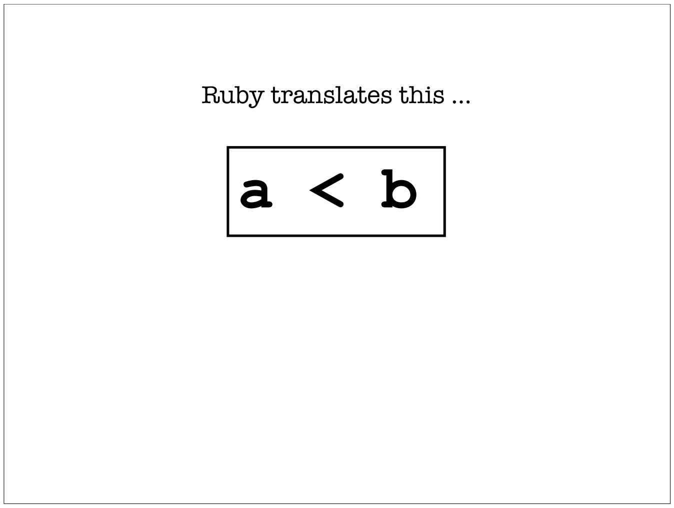


```
>> require 'book'
=> true
>> a = Book.new("Learning to Program",
                "Chris Pine")
=> #<...>
>> b = Book.new("Godel, Escher, Bach",
                "Douglas Hofstedter")
=> #<...>
>> a < b
NoMethodError: undefined method '<'
     for #<Book:0x72b14>
    from (irb):4
```



:< is a method!

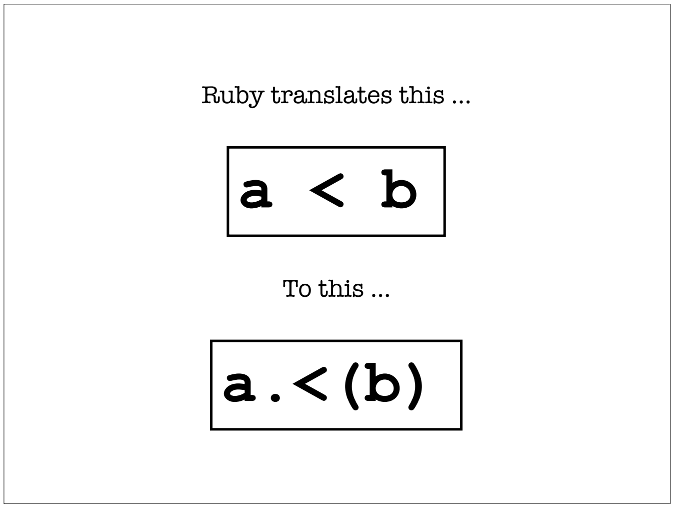
```
>> require 'book'
=> true
>> a = Book.new("Learning to Program",
                "Chris Pine")
=> #<...>
>> b = Book.new("Godel, Escher, Bach",
                "Douglas Hofstedter")
=> #<...>
>> a < b
NoMethodError: undefined method '<'
     for #<Book:0x72b14>
    from (irb):4
```



Ruby translates this ...

```
a < b
```

337



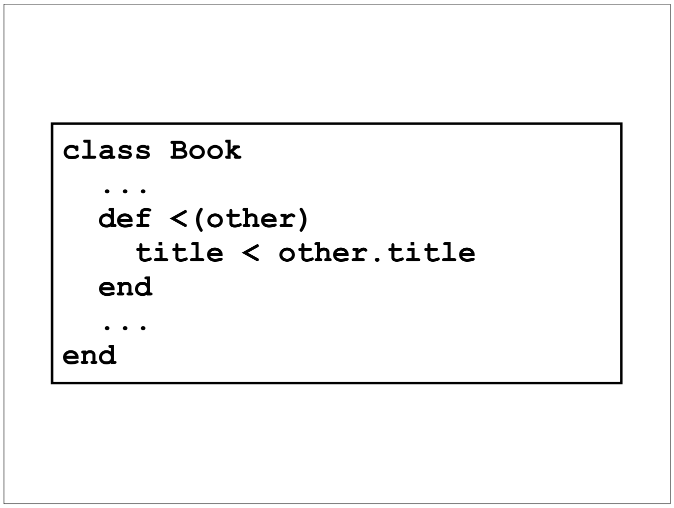
Ruby translates this ...

```
a < b
```

To this ...

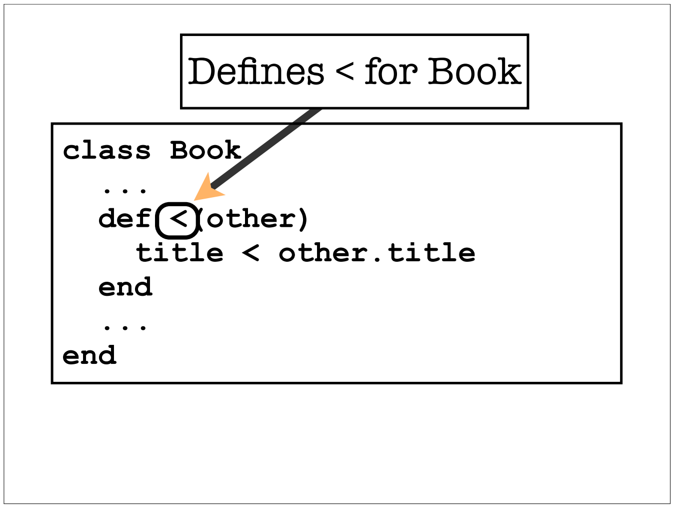
```
a.<(b)
```

338



```
class Book
  ...
  def <(other)
    title < other.title
  end
  ...
end
```

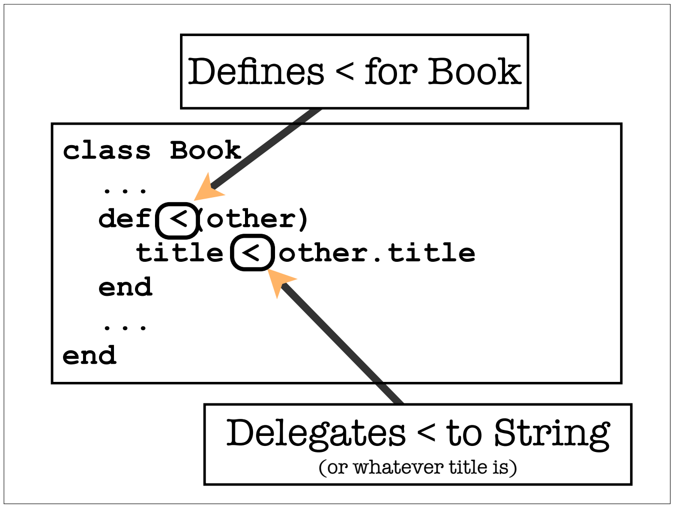
339



Defines < for Book

```
class Book
  ...
  def <(other)
    title < other.title
  end
  ...
end
```

340

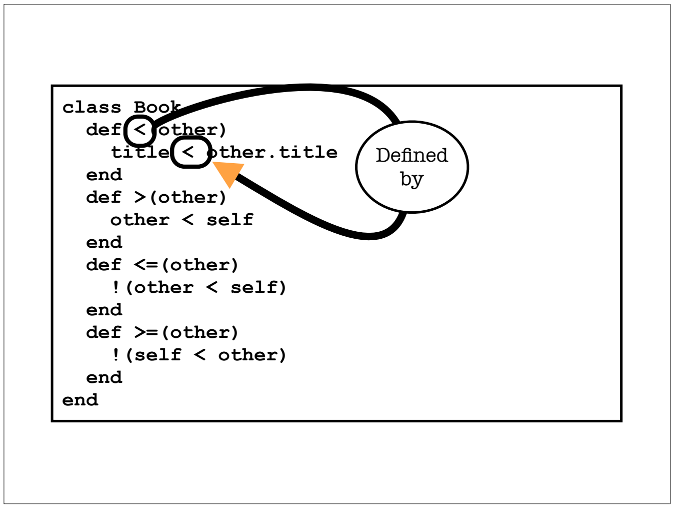


Defines < for Book

```
class Book
  ...
  def <(other)
    title < other.title
  end
  ...
end
```

Delegates < to String
(or whatever title is)

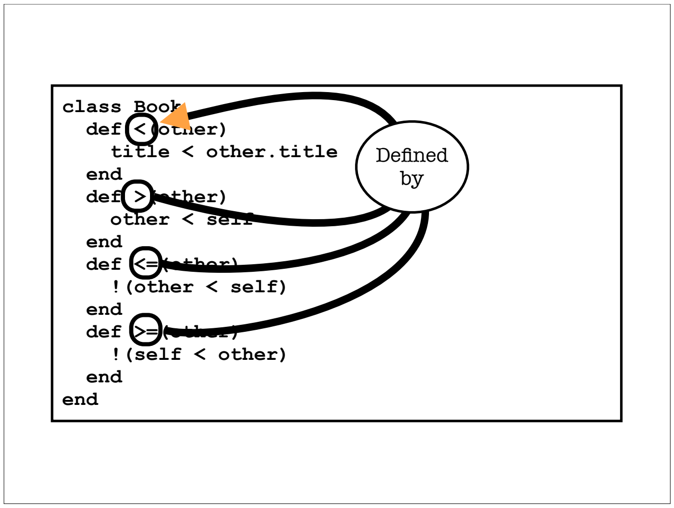
341



```
class Book
  def <(other)
    title < other.title
  end
  def >(other)
    other < self
  end
  def <=(other)
    !(other < self)
  end
  def >=(other)
    !(self < other)
  end
end
```

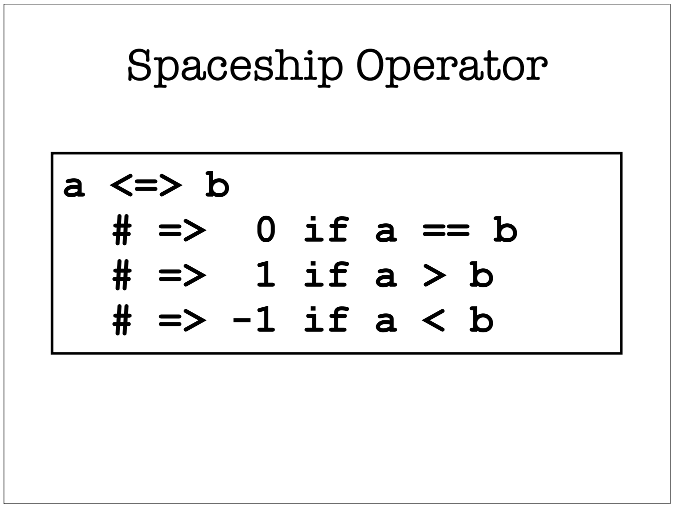
Defined by

342



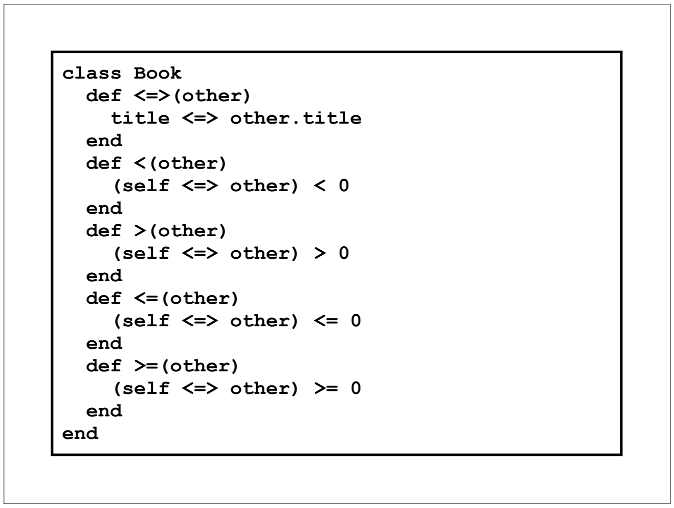
```
class Book
  def <(other)
    title < other.title
  end
  def >(other)
    other < self
  end
  def <=(other)
    !(other < self)
  end
  def >=(other)
    !(self < other)
  end
end
```

Defined by

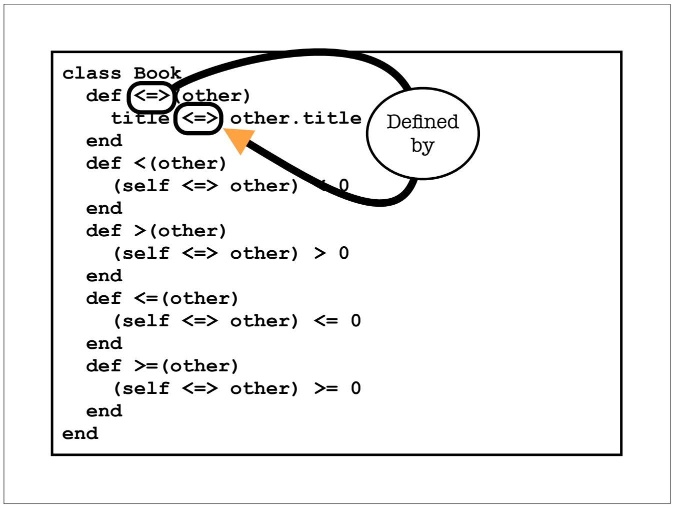


Spaceship Operator

```
a <=> b
  # =>  0 if a == b
  # =>  1 if a > b
  # => -1 if a < b
```

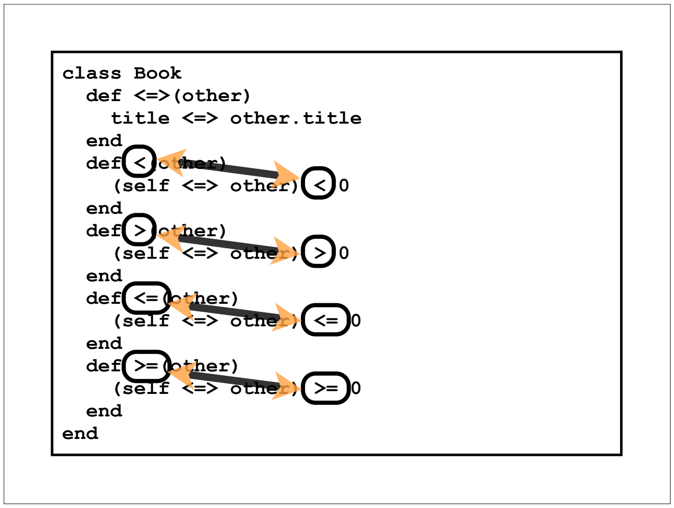


```
class Book
  def <=>(other)
    title <=> other.title
  end
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
end
```

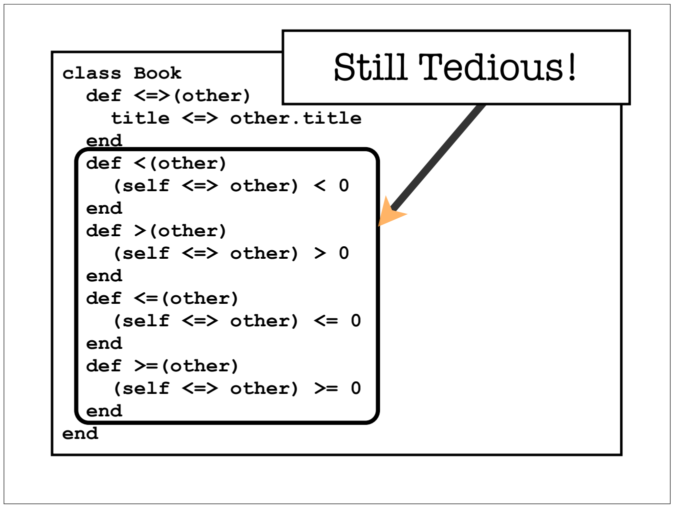


```
class Book
  def <=>(other)
    title <=> other.title
  end
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
end
```

Defined by

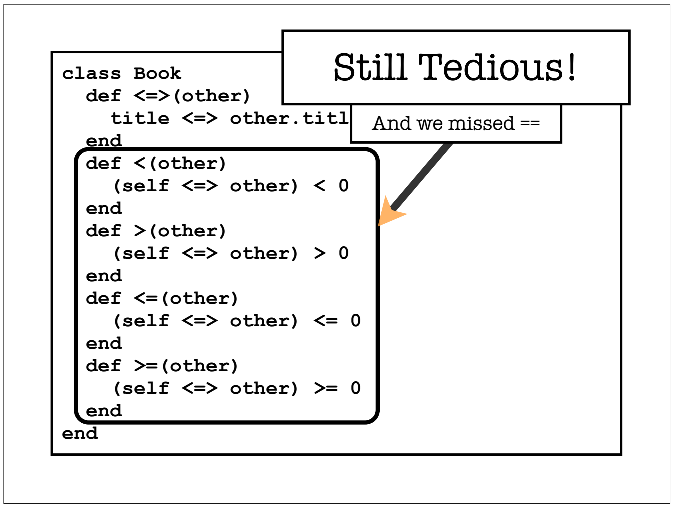


```
class Book
  def <=>(other)
    title <=> other.title
  end
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
end
```



```
class Book
  def <=>(other)
    title <=> other.title
  end
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
end
```

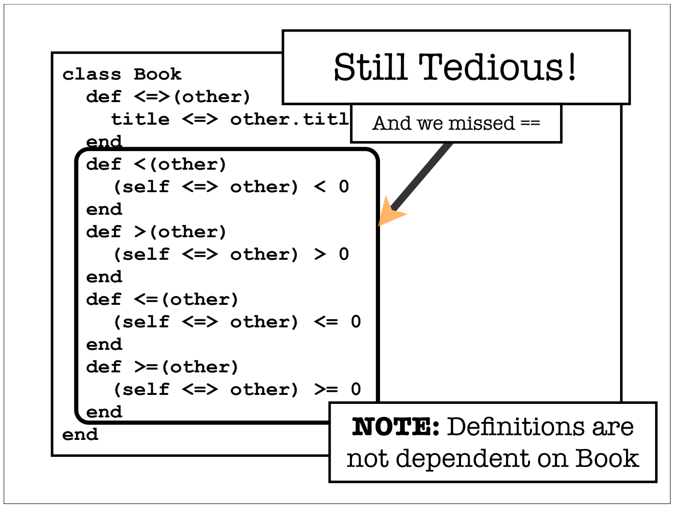
Still Tedious!



```
class Book
  def <=>(other)
    title <=> other.titl
  end
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
end
```

Still Tedious!

And we missed ==

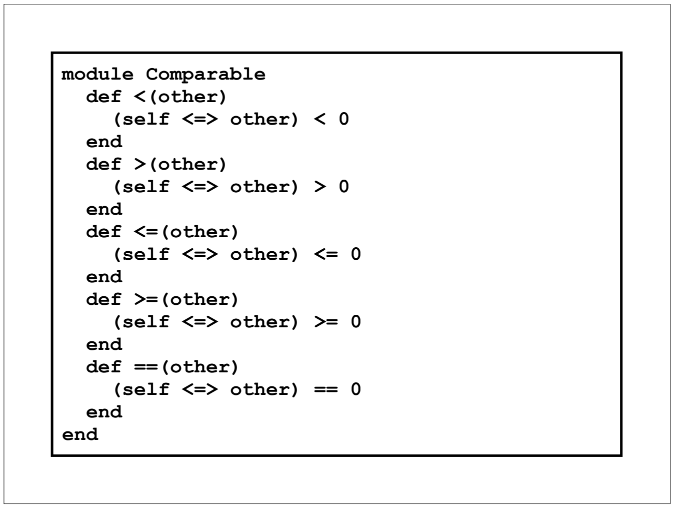


```
class Book
  def <=>(other)
    title <=> other.titl
  end
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
end
```

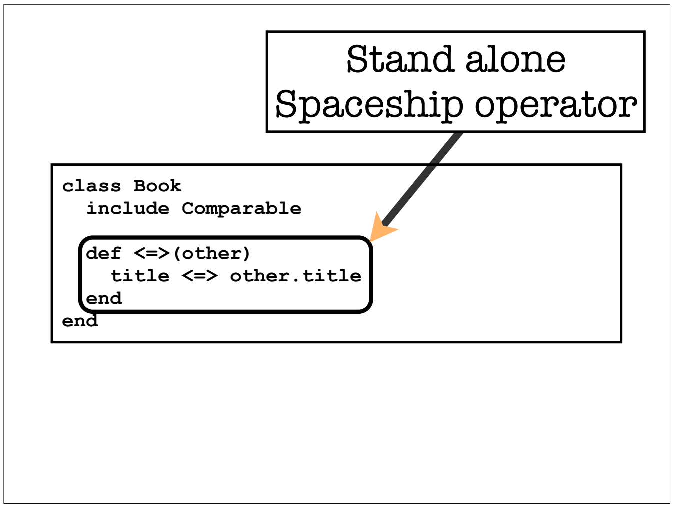
Still Tedious!

And we missed ==

NOTE: Definitions are not dependent on Book

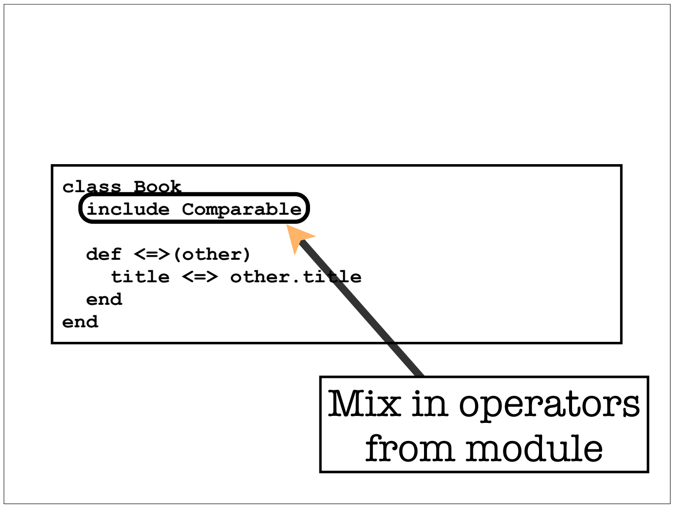


```
module Comparable
  def <(other)
    (self <=> other) < 0
  end
  def >(other)
    (self <=> other) > 0
  end
  def <=(other)
    (self <=> other) <= 0
  end
  def >=(other)
    (self <=> other) >= 0
  end
  def ==(other)
    (self <=> other) == 0
  end
end
```



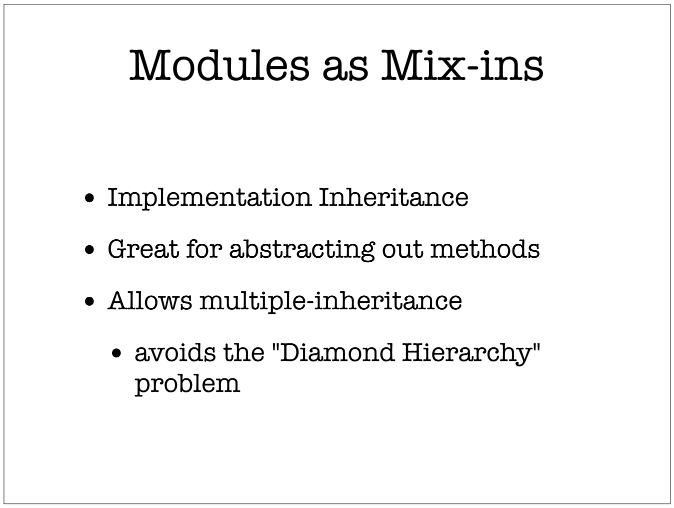
Stand alone Spaceship operator

```
class Book
  include Comparable

  def <=>(other)
    title <=> other.title
  end
end
```



```
class Book
  include Comparable

  def <=>(other)
    title <=> other.title
  end
end
```

Mix in operators from module



Modules as Mix-ins

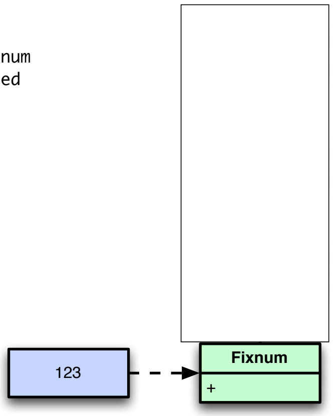
- Implementation Inheritance
- Great for abstracting out methods
- Allows multiple-inheritance
  - avoids the "Diamond Hierarchy" problem

Class Environment

Method Lookup

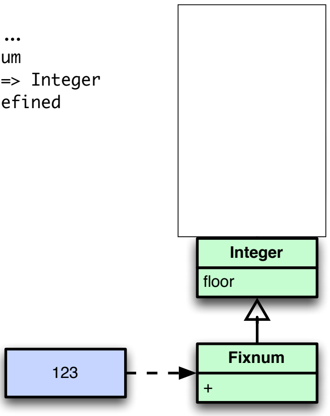
Evaluating 123+1 ...

```
123.class   #=> Fixnum
Fixnum has :+ defined
Invoke Fixnum#+
```



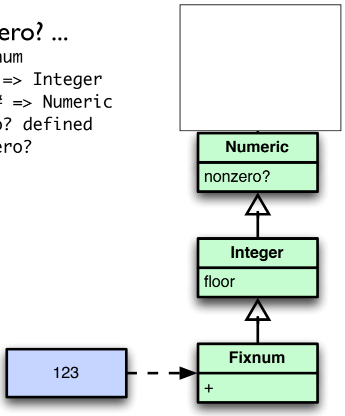
Evaluating 123.floor ...

```
123.class   #=> Fixnum
Fixnum.superclass # => Integer
Integer has :floor defined
Invoke Integer#floor
```



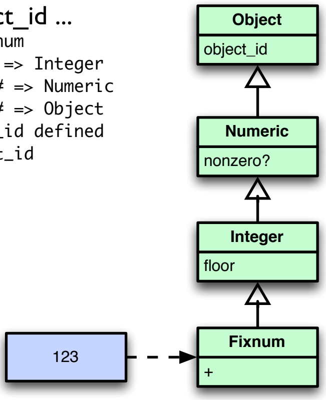
Evaluating 123.nonzero? ...

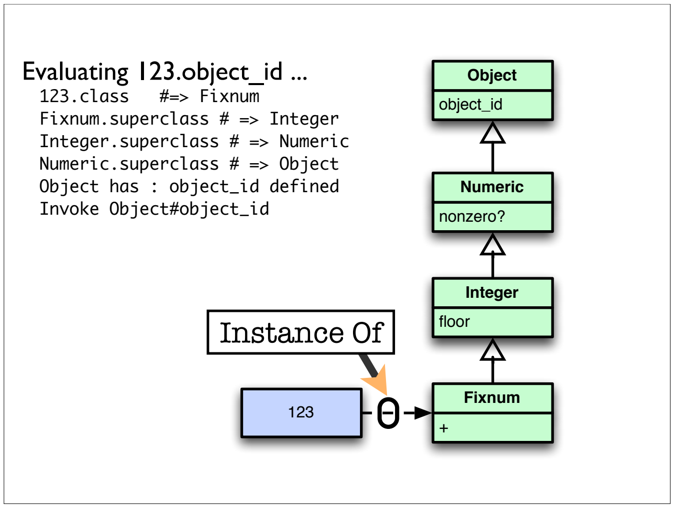
```
123.class   #=> Fixnum
Fixnum.superclass # => Integer
Integer.superclass # => Numeric
Numeric has :nonzero? defined
Invoke Numeric#nonzero?
```



Evaluating 123.object_id ...

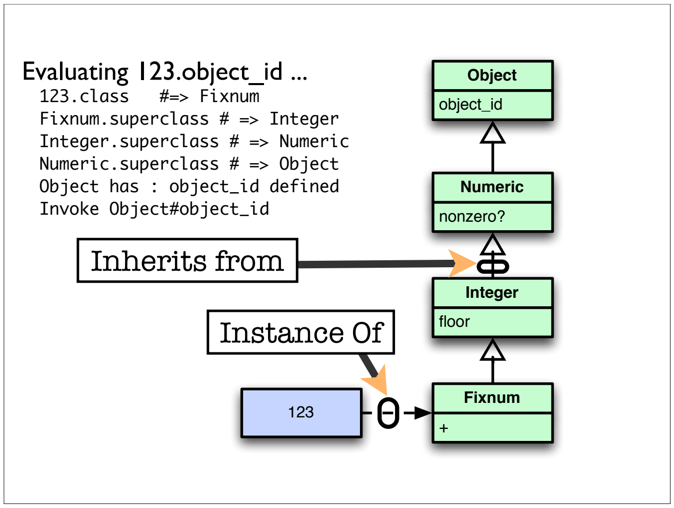
```
123.class   #=> Fixnum
Fixnum.superclass # => Integer
Integer.superclass # => Numeric
Numeric.superclass # => Object
Object has : object_id defined
Invoke Object#object_id
```





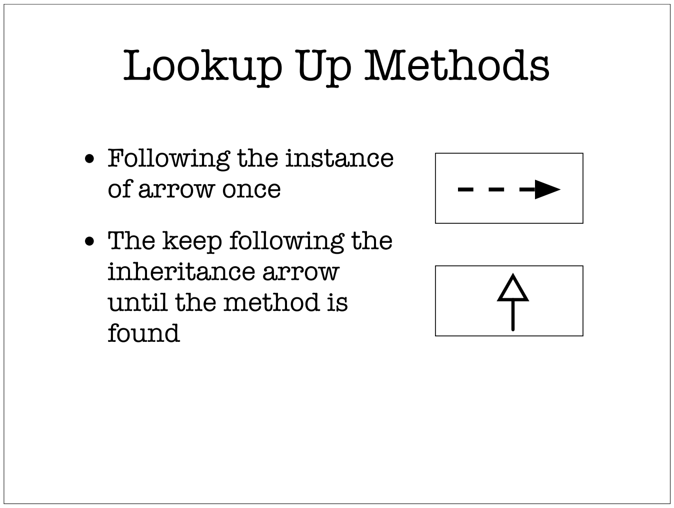
Evaluating 123.object_id ...

```
123.class    #=> Fixnum
Fixnum.superclass # => Integer
Integer.superclass # => Numeric
Numeric.superclass # => Object
Object has : object_id defined
Invoke Object#object_id
```



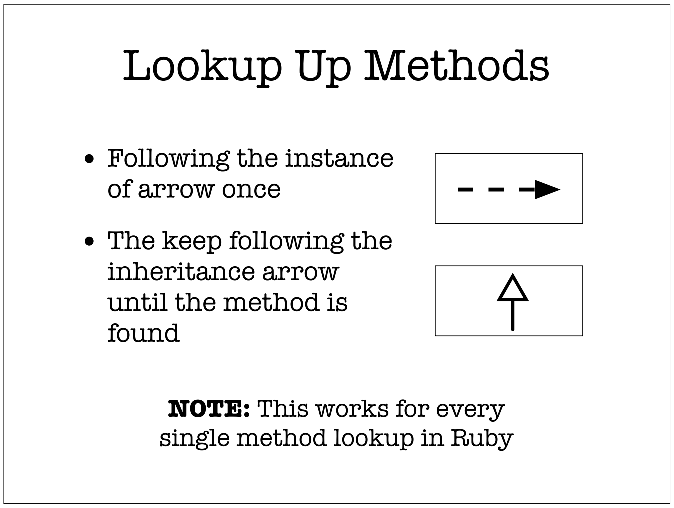
Evaluating 123.object_id ...

```
123.class    #=> Fixnum
Fixnum.superclass # => Integer
Integer.superclass # => Numeric
Numeric.superclass # => Object
Object has : object_id defined
Invoke Object#object_id
```



Lookup Up Methods

- Following the instance of arrow once
- The keep following the inheritance arrow until the method is found

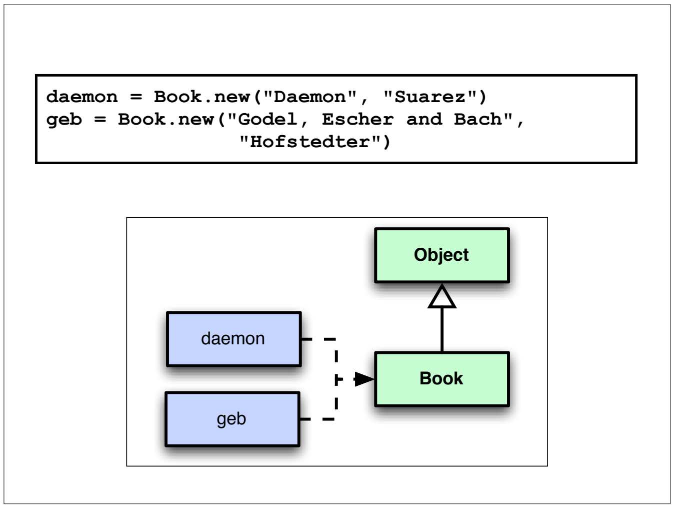


Lookup Up Methods

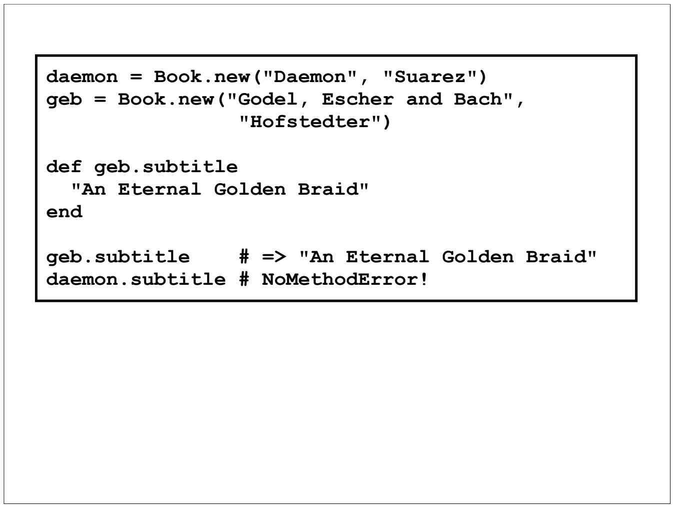
- Following the instance of arrow once
- The keep following the inheritance arrow until the method is found

NOTE: This works for every single method lookup in Ruby

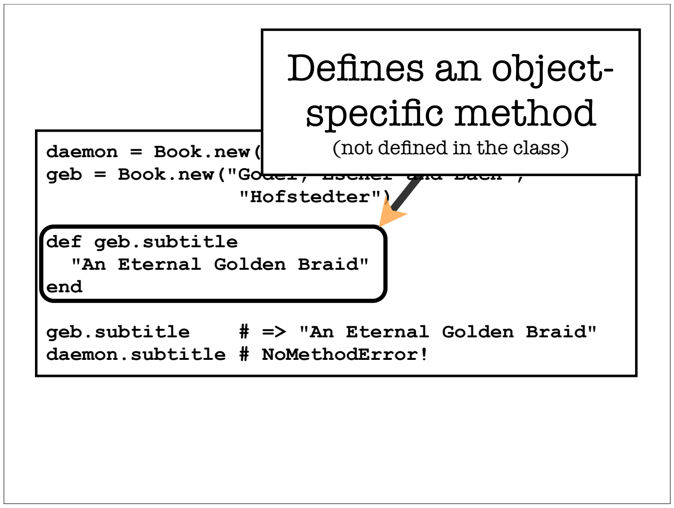
Singleton Methods



```
daemon = Book.new("Daemon", "Suarez")
geb = Book.new("Godel, Escher and Bach",
               "Hofstedter")
```



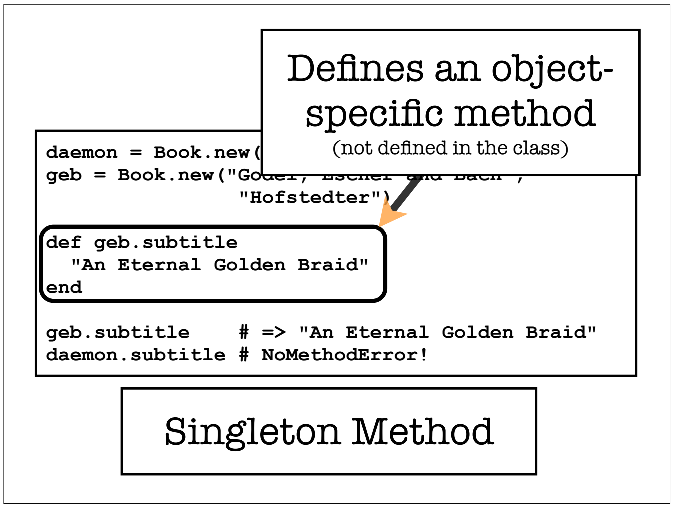
```
daemon = Book.new("Daemon", "Suarez")
geb = Book.new("Godel, Escher and Bach",
               "Hofstedter")

def geb.subtitle
  "An Eternal Golden Braid"
end

geb.subtitle    # => "An Eternal Golden Braid"
daemon.subtitle # NoMethodError!
```



Defines an object-specific method

(not defined in the class)

```
daemon = Book.new("Daemon", "Suarez")
geb = Book.new("Godel, Escher and Bach",
               "Hofstedter")

def geb.subtitle
  "An Eternal Golden Braid"
end

geb.subtitle    # => "An Eternal Golden Braid"
daemon.subtitle # NoMethodError!
```

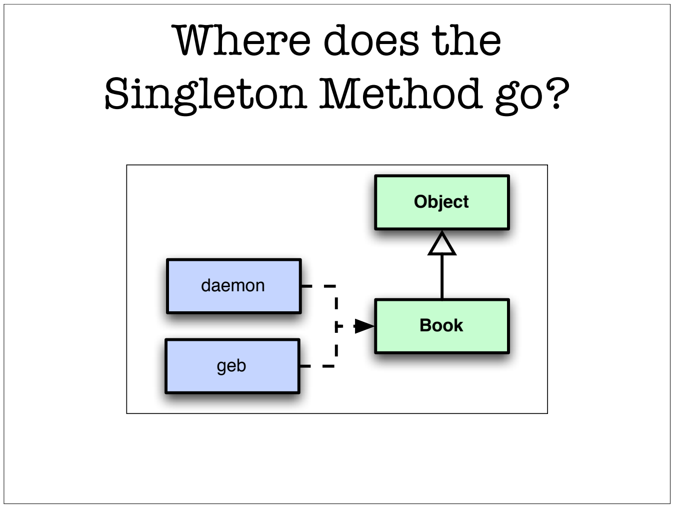


Defines an object-specific method

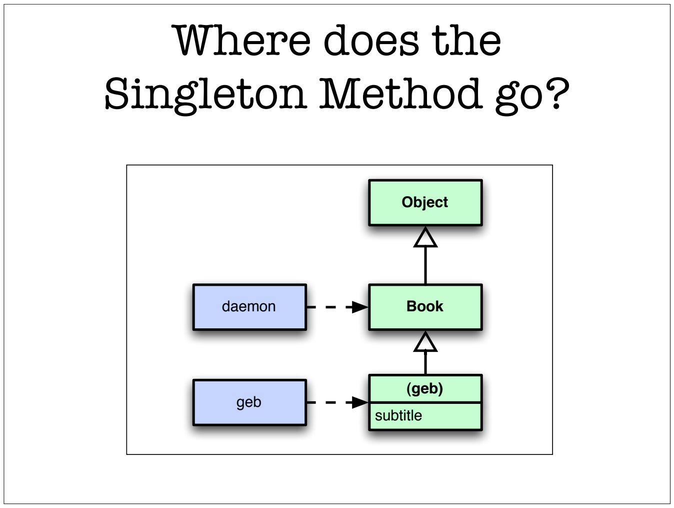
(not defined in the class)

```
daemon = Book.new("Daemon", "Suarez")
geb = Book.new("Godel, Escher and Bach",
               "Hofstedter")

def geb.subtitle
  "An Eternal Golden Braid"
end

geb.subtitle    # => "An Eternal Golden Braid"
daemon.subtitle # NoMethodError!
```

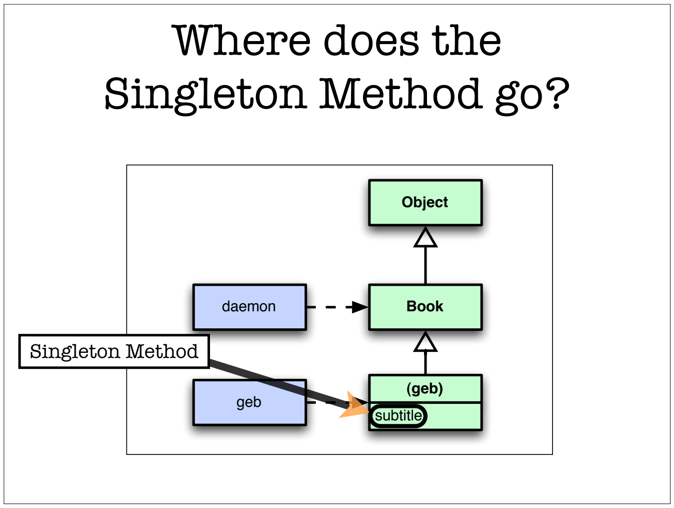
Singleton Method



Where does the Singleton Method go?



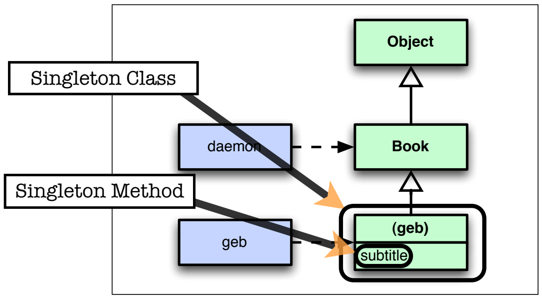
Where does the Singleton Method go?



Where does the Singleton Method go?

Singleton Method

Where does the Singleton Method go?

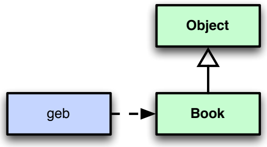


Singleton Classes

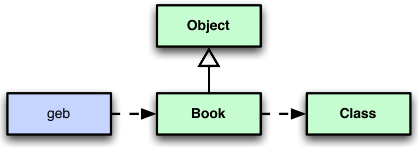
- Created as needed, on demand
- Per object, never shared 'tween objects
- Nearly Invisible
  - obj.class still returns original class
  - implementation detail
- Other Names
  - Metaclass (inaccurate)
  - Eigenclass

Class Methods

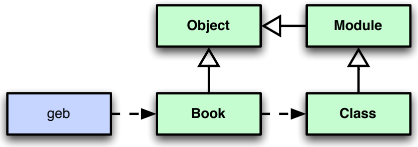
```
>> daemon = Book.new("Daemon", "Suarez")
=> #<...>
>> daemon.class
=> Book
>>
```

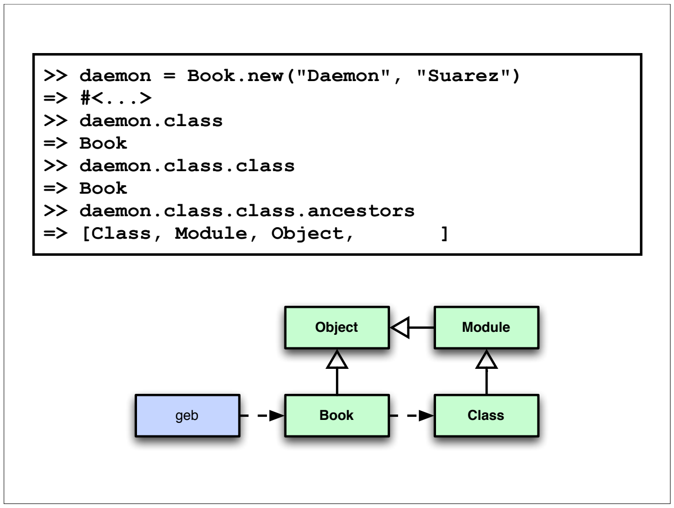


```
>> daemon = Book.new("Daemon", "Suarez")
=> #<...>
>> daemon.class
=> Book
>> daemon.class.class
=> Class
```

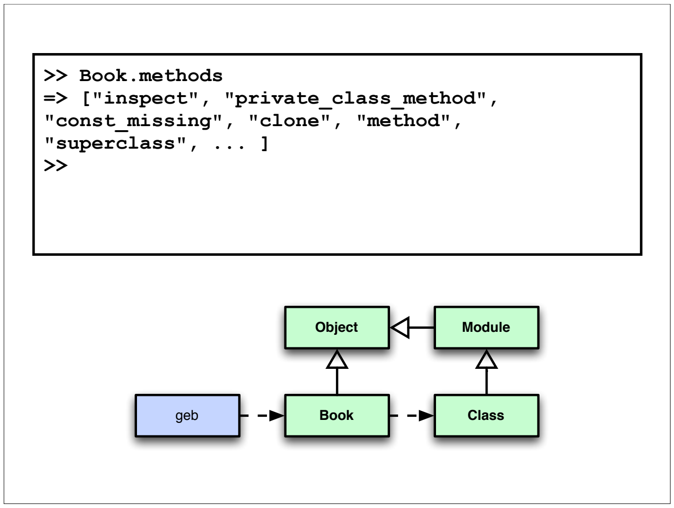


```
>> daemon = Book.new("Daemon", "Suarez")
=> #<...>
>> daemon.class
=> Book
>> daemon.class.class
=> Book
>> daemon.class.class.ancestors
=> [Class, Module, Object, Kernel]
```

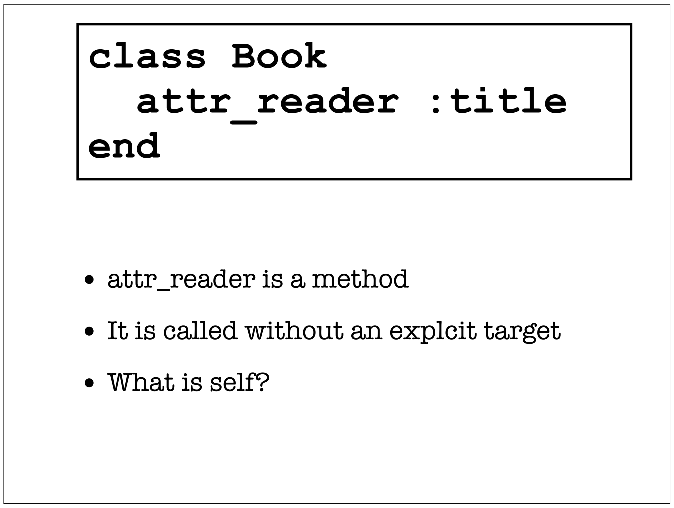




```
>> daemon = Book.new("Daemon", "Suarez")
=> #<...>
>> daemon.class
=> Book
>> daemon.class.class
=> Book
>> daemon.class.class.ancestors
=> [Class, Module, Object,       ]
```

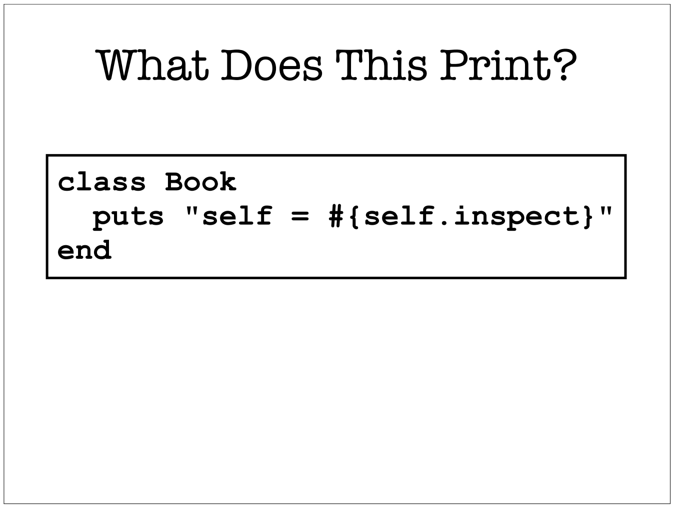


```
>> Book.methods
=> ["inspect", "private_class_method",
"const_missing", "clone", "method",
"superclass", ... ]
>>
```



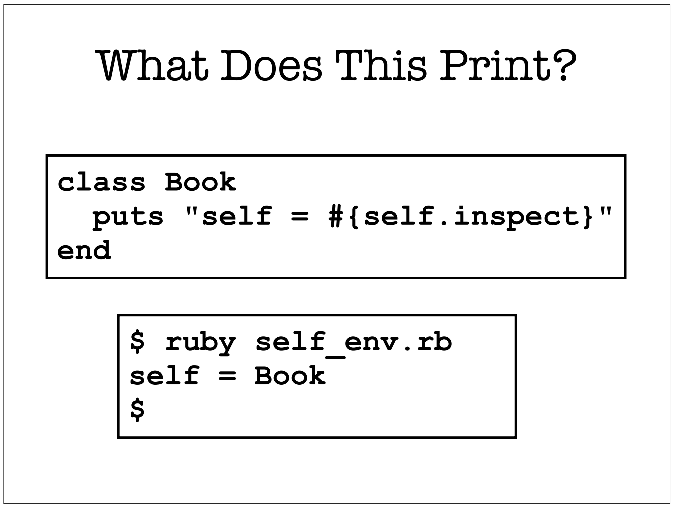
```
class Book
  attr_reader :title
end
```

- attr_reader is a method
- It is called without an explcit target
- What is self?



What Does This Print?

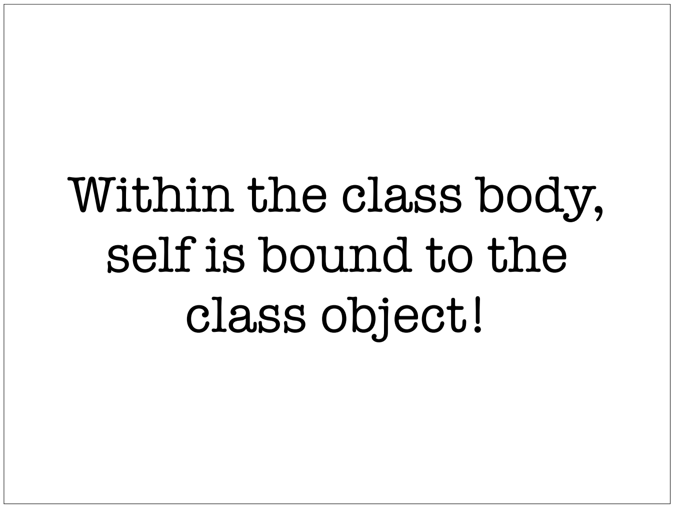
```
class Book
  puts "self = #{self.inspect}"
end
```



What Does This Print?

```
class Book
  puts "self = #{self.inspect}"
end
```

```
$ ruby self_env.rb
self = Book
$
```



Within the class body, self is bound to the class object!

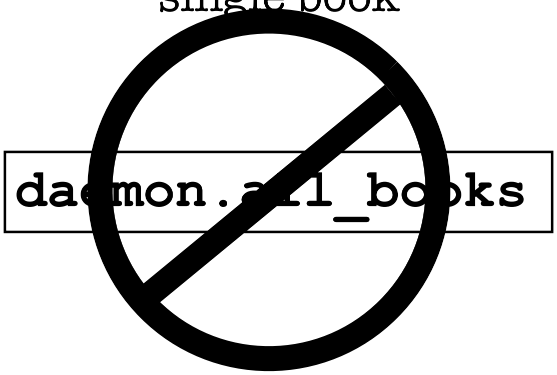
Keep this in mind, just for a bit ...

New Book Requirements

- Need to keep track of **all** books created.
- The method "all_books" will return an array of all books created

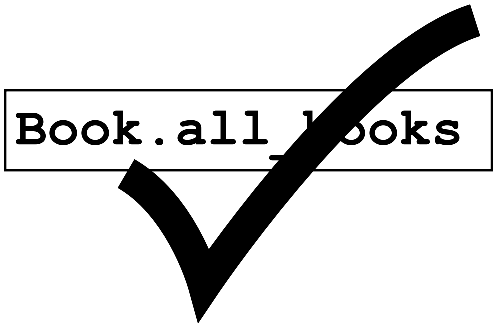
- Question: Where should "all_books" go?

Not a property of a single book

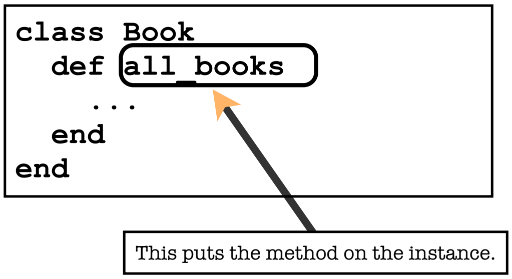


```
daemon.all_books
```

Property of the Book Class

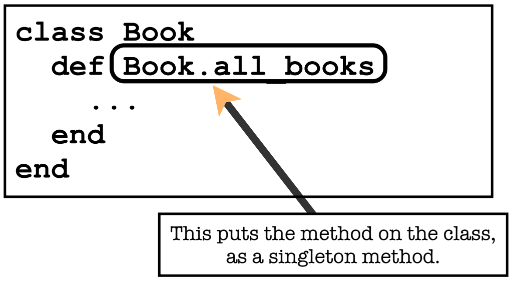


```
Book.all_books
```



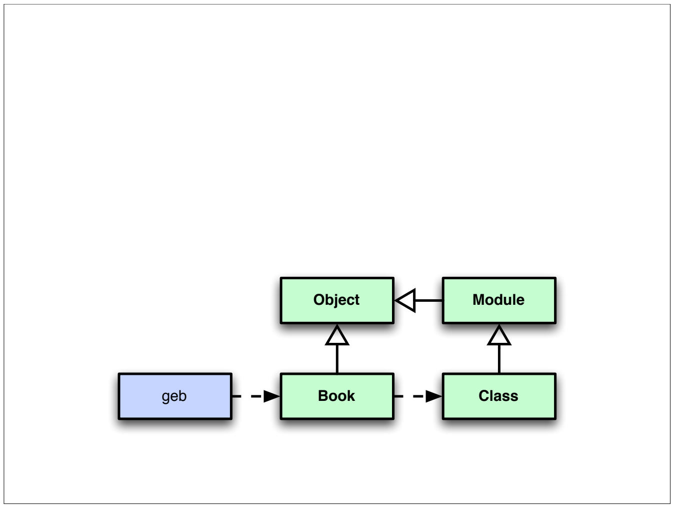
```
class Book
  def all_books
    ...
  end
end
```

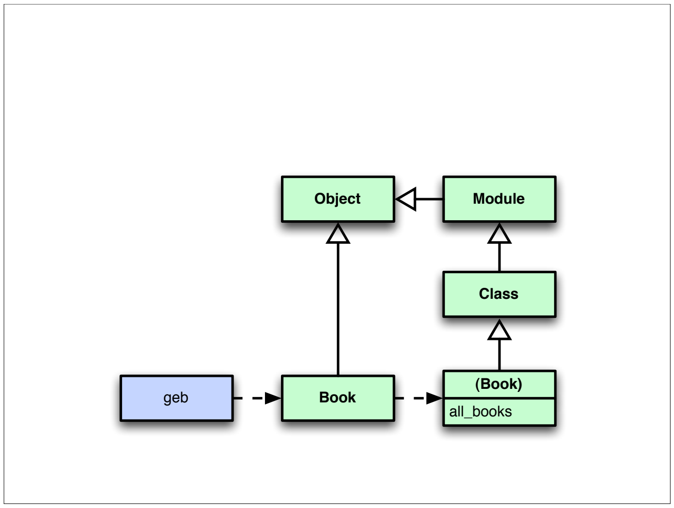
This puts the method on the instance.

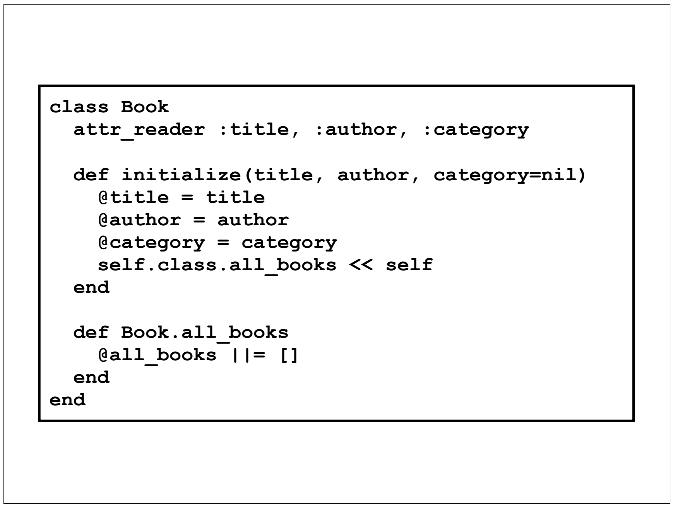


```
class Book
  def Book.all_books
    ...
  end
end
```

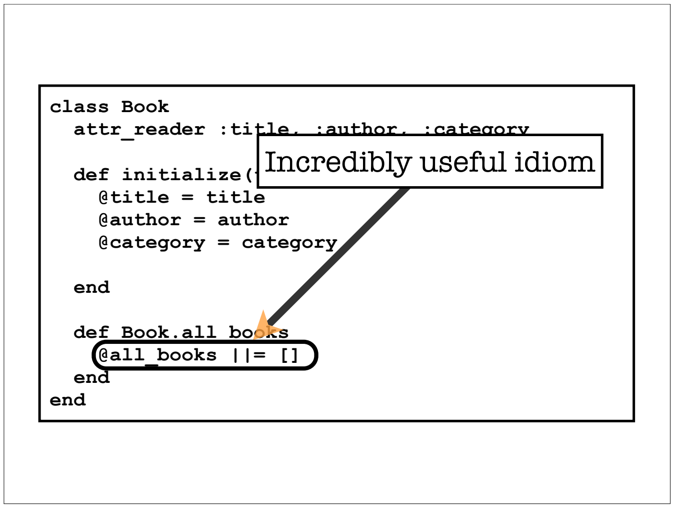
This puts the method on the class, as a singleton method.





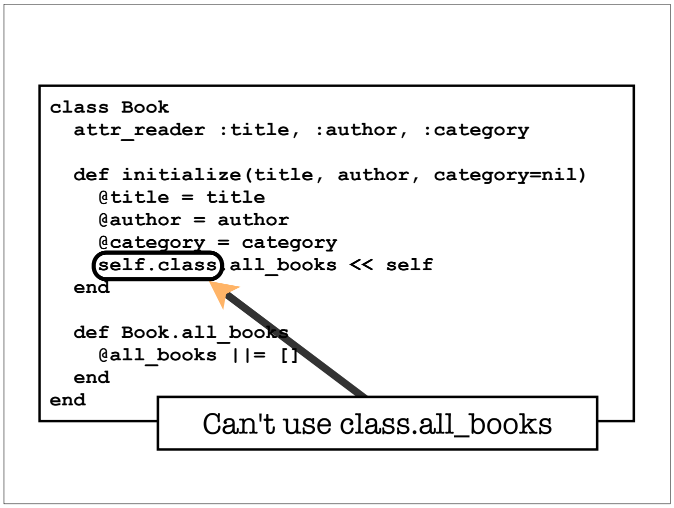


```
class Book
  attr_reader :title, :author, :category

  def initialize(title, author, category=nil)
    @title = title
    @author = author
    @category = category
    self.class.all_books << self
  end

  def Book.all_books
    @all_books ||= []
  end
end
```



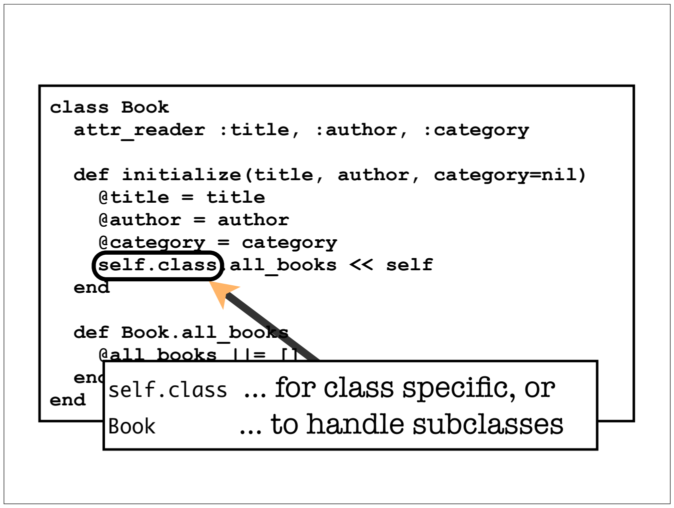
```
class Book
  attr_reader :title, :author, :category

  def initialize(
    @title = title
    @author = author
    @category = category

  end

  def Book.all_books
    @all_books ||= []
  end
end
```

Incredibly useful idiom



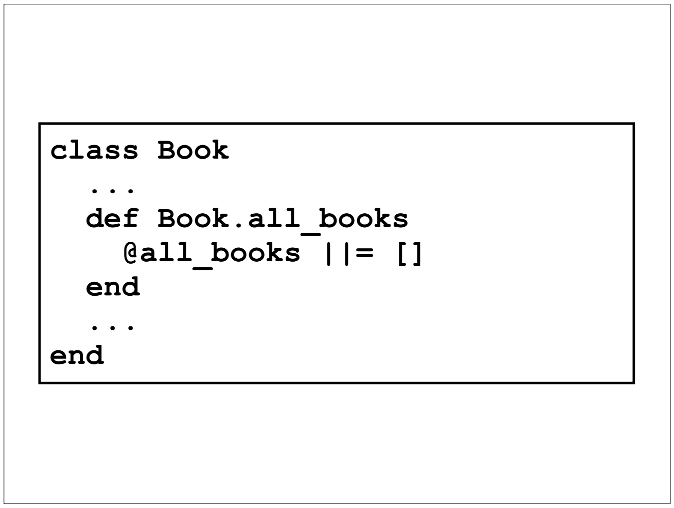
```
class Book
  attr_reader :title, :author, :category

  def initialize(title, author, category=nil)
    @title = title
    @author = author
    @category = category
    self.class.all_books << self
  end

  def Book.all_books
    @all_books ||= []
  end
end
```

Can't use class.all_books

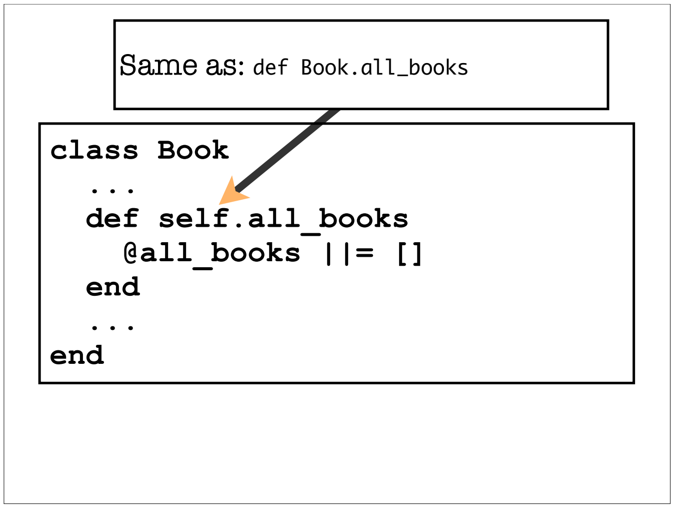


```
class Book
  attr_reader :title, :author, :category

  def initialize(title, author, category=nil)
    @title = title
    @author = author
    @category = category
    self.class.all_books << self
  end

  def Book.all_books
    @all_books ||= []
  end
end
```

`self.class` ... for class specific, or
`Book` ... to handle subclasses

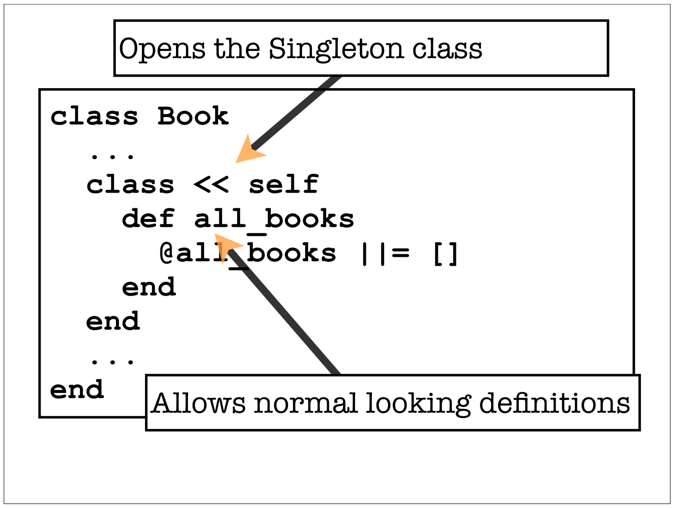


```
class Book
  ...
  def Book.all_books
    @all_books ||= []
  end
  ...
end
```



Same as: `def Book.all_books`

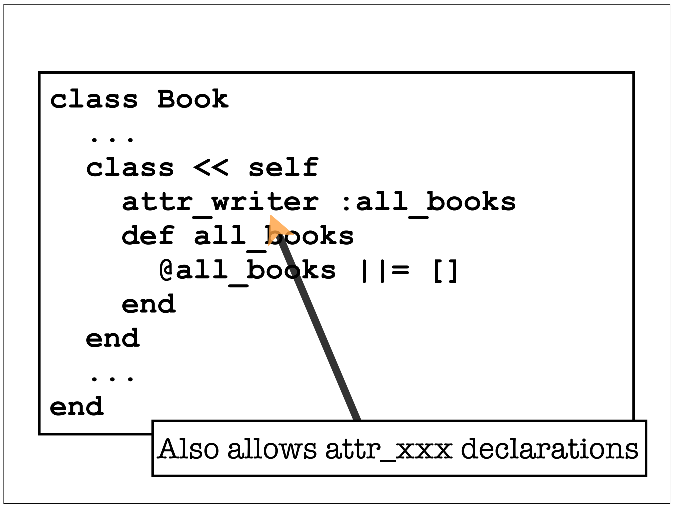
```
class Book
  ...
  def self.all_books
    @all_books ||= []
  end
  ...
end
```



Opens the Singleton class

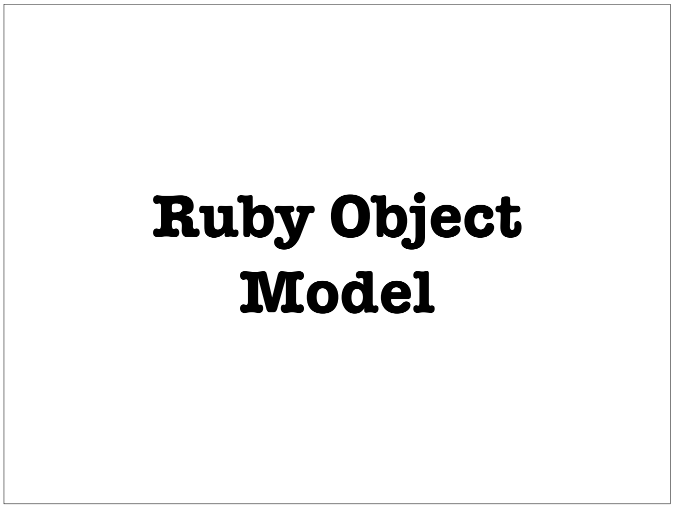
```
class Book
  ...
  class << self
    def all_books
      @all_books ||= []
    end
  end
  ...
end
```

Allows normal looking definitions



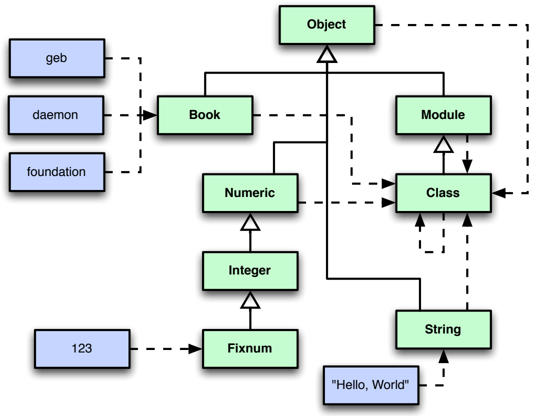
```
class Book
  ...
  class << self
    attr_writer :all_books
    def all_books
      @all_books ||= []
    end
  end
  ...
end
```

Also allows attr_xxx declarations

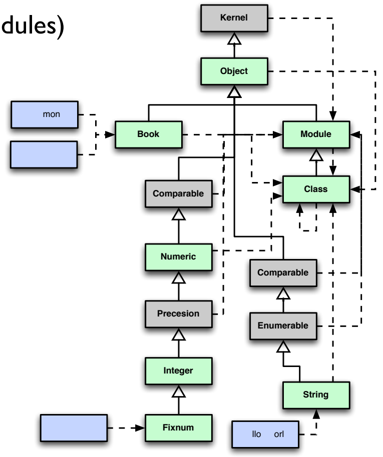


Ruby Object Model

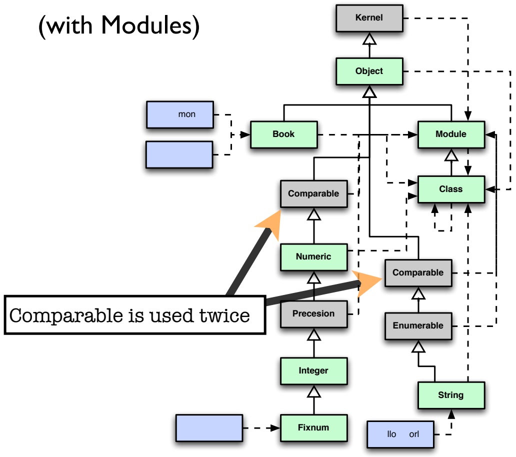
(simplified)



(with Modules)



(with Modules)



Comparable is used twice

LAB 6

TBD

Project A

Conference Scheduling

Project B

20-Questions

Project C

Sudoku Solver