Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Caffeinated Crash Course in Ruby

Ben Weissmann

With help from Mason Glidden
Sponsored by Fluid Interfaces

January 28, 2013

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Me

- Ben Weissmann
- bsw@mit.edu
- Ruby, Rails for 6 years
- Build Ruby/Rails apps at Twitter, TripAdvisor, Fluid Interfaces

This Class

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

- Super-fast-paced crash course
- Excercise-based
- Resources:
  http://www.ruby-doc.org/core
  https://rubyclass.meteor.com/
  http://tinyurl.com/rubyslides

Inspirations & Motivation

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

- Released in 1995 by Yukihiro "matz" Matsumoto
- Influenced by Perl, Smalltalk, Eiffel, Ada, and Lisp.
- Principle of Least Surprise
- Quotes from matz:
  - "Ruby is designed to make programmers happy."
  - "I knew alot about the language's target audience: myself."
  - "trying to make Ruby natural, not simple"
  - "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python."

Hello World

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
puts "Hello MIT"
```

- `puts` is a method in the `Kernel` module.
- `"Hello MIT"` is a string.
- Parentheses are optional. We could have done:

```
puts("Hello MIT")
```

Using Ruby

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

- `irb`: Ruby's REPL
- `ruby`: Run ruby scripts
- Follow along: run hello world with irb and ruby.

Math

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

- Very similar to most other languages
- Floating point vs integers: try `5/2` and `5.0/2` in irb
- Other stuff: +, -, *, /, ** (exponentiation), % (modulus).

Variables

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
num_cats = 10
num_dogs = 20
num_pets = num_cats + num_dogs # => 30
too_many_pets = 1_000_000
PET_LIMIT = 123
```

- Dynamic typing
- underscores, not camelCase
- Initial capital means a constant (convention: ALL_CAPS)
- BTW, use a # for comments. You can use `=begin` and `=end` for multiline comments, but it's not really used.

Assignment

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Assignment returns a value

```
10 + (ham_sandwich = 10)    # => 20
ham_sandwich                # => 10
```

Multiple assignment is done in parallel

```
ham, cheese = 10, 20
ham           # => 10
cheese        # => 20
ham, cheese = cheese, ham
ham           # => 20
cheese        # => 10
```

Assignment operators

```
tequila_shots = 5
tequila_shots += 3
tequila_shots # => 8
```

true, false, nil

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

`true`	Represents boolean true. Instance of TrueClass
`false`	Represents boolean False. Instance of FaleClass
`nil`	Represents no value. Instance of NilClass

Strings

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

- Concatenate with +

```
"ruby" + "class" # => "rubyclass"
```

- Interpolate with #{}

```
action = "give you up"
opposite = "never gonna #{my_str}"
# => "never gonna give you up"
```

Method Invocation

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
"green".reverse                 # => "neerg"
"123".to_i                      # => 123
"green".upcase                  # => "GREEN"
fav_color = "  green  "         # => "  green  "
fav_color.strip                 # => "green"
fav_color                       # => "  green  "
fav_color.strip!                # => "green"
fav_color                       # => "green"
fav_color.starts_with? "gr"     # => true
"green"[1..-2]                  # => "ree"
```

- Use a `.` to invoke a method on an object.
- All methods return a value. That value might be `nil`
- `?` means it returns a boolean. `!` means it's destructive.

User Input and the If Statement

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
puts "Shall I get you a sandwich?"
response = gets.strip.downcase
if response == "yes"
    puts "No way."
elsif response == "sudo make me a sandwich"
    puts "OK."
else
    puts "Why don't you want a sandwich?"
end
```

- `gets` gets a line of input, including a newline
- `==` is equality. Also: `!=`, `>`, `<`, etc.
- `if`, `elsif`, `else`. No parens required.

Exercise: The Stringinator

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Create a program that gets a command and a string from the user, and then performs that command on the string. You should support `upcase`, `downcase`, and `reverse`. Example:

```
Welcome to The Stringinator.  Enter a string:
foobar
Enter a command:
upcase
Your new string is:  FOOBAR
```

http://rubyclass.meteor.com

Solution: The Stringinator

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
puts "Welcome to the Stringinator. Enter a string:"
str = gets.strip
puts "Enter a command:"
cmd = gets.strip.downcase

if cmd == "upcase"
    out = str.upcase
elsif cmd == "downcase"
    out = str.downcase
elsif cmd == "reverse"
    out = str.reverse
else
    out = "i don't understand you"
end

puts "Your new string is: #{out}"
```

A Better Solution

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
puts "Welcome to the Stringinator. Enter a string:"
str = gets.strip
puts "Enter a command:"
cmd = gets.strip.downcase

out = case cmd
      when "upcase" then str.upcase
      when "downcase" then str.downcase
      when "reverse"
        str.reverse
      else "i don't understand you"
      end

puts "Your new string is: #{out}"
```

More Comparators

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
(a && b)                    == (a and b)
(a || b)                    == (a or b)
((a == b) or (a > b))       == (a >= b)
((a == b) or (a < b))       == (a <= b)
 a && b  or  c && d         == (a && b) or (c && d)
```

Arrays

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
prices = [10, 20, 30] # => [10, 20, 30]
prices + [30, 40]     # => [10, 20, 30, 40, 50]
prices                # => [10, 20, 30]
prices.push 40        # => [10, 20, 30, 40]
prices                # => [10, 20, 30, 40]
prices.pop            # => 40
prices                # => [10, 20, 30]
prices[0]             # => 10
prices[-1]            # => 30
prices[0..1]          # => [10, 20]
prices.length         # => 3
prices.reverse        # => [30, 20, 10]
```

Assignment with Arrays

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Ruby unpacks arrays for you:

```
a, b, c = [1, 2, 3]
a       # => 1
b       # => 2
c       # => 3

d, e = [4, 5, 6]
d    # => 4
e    # => 5

f, g, h, i = [7, 8, 9]
f          # => 7
g          # => 8
h          # => 9
i          # => nil
```

Splats

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

*, the splat operator, packs/unpacks arrays:

```
a, b, *c = 1, 2, 3, 4, 5
a # =>
 1
b # => 2
c # => [3, 4, 5]

q, r, s = 10, 20, *[30, 40, 50]
q # => 10
r # => 20
s # => 30
```

Iterators

One way to iterate over a list:

```
cheeses = ["swiss", "brie", "cheddar"]
for cheese in cheeses
  puts cheese
end
```

The more ruby-ish way:

```
cheeses = ["swiss", "brie", "cheddar"]
cheeses.each do |cheese|
  puts cheese
end
```

While

One way to iterate over a list:

```
while rabbit_is_hungry
  feed_rabbit_a_carrot
end
```

Exploit that assignment returns a value:

```
while (input = gets.strip) != "STOP"
  puts "You said #{input}"
end
```

Excercise: Guessing Game

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Make a guessing game that tell the user over/under and how far away their guesses were. `rand(n)` returns a number between 0 and n-1. `my_str.to_i` converts a string to an integer.

```
Make a guess:
10
Too High
5
Too Low
7
Correct!  10 was 3 away.  5 was 2 away.  7 was 0
away.
```

Solution: Guessing Game

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
guesses = []
correct = rand(20)
puts "Make a guess:"
while (guess = gets.strip.to_i) != correct
  guesses.push guess
  if guess > correct
    puts "Too high"
  else
    puts "Too low"
  end
end
puts "Correct!"
guesses.each do |guess|
  puts "#{guess} was #{(guess - correct).abs} away"
end
```

More Iterators

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Ruby supports lots of iterators:

```ruby
a = [1,2,3,4,5]
a.select{|x| x % 2 == 0} # => [2, 4]

b = ["10", "21", "32"]
b.map do |x|
  x = x.to_i
  if x % 2 == 0
    x / 2
  else
    3*x + 1
  end
end
# => [5, 64, 16]
```

Other Methods That Take Blocks

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
3.times do |i|
  puts i
end

["red", "green", "blue"].each_with_index do |clr, i|
    puts "color #{i} was #{clr}"
end

[0, 10, 20].map do |item, idx|
    item * 2
end
# => [0, 20, 40]

Array.new(5) do |idx|
  idx * 5
end
```

Blocks with Multiple Arguments

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

There are two ways to give multiple arguments to a block: by yielding multiple arguments (like `each_with_index`) or by yielding a single array (which ruby will unpack)

```ruby
["a", "b", "c"].each_with_index do |c, i|
  puts "letter #{i} is #{c}"
end
# => "letter 0 is a", etc.

["a", "b", "c"].each_with_index do |c|
  puts "the letter is #{c}"
end
# => "the letter is a", etc.

["a", "b", "c"].each_with_index do |*pair|
  puts "letter #{pair[1]} is #{pair[0]}"
end
# => "letter 0 is a", etc.
```

Blocks with Multiple Arguments, Cont.

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
[["a", "i"], ["b", "ii"]].each do |c, r|
  puts "letter #{c}, numeral #{r}"
end
# => "letter a, numeral i", etc.

[["a", "i"], ["b", "ii"]].each do |pair|
  puts "letter #{pair[0]}, numeral #{pair[1]}"
end
# => "letter a, numeral i", etc.
```

Chaning Enumerators

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
[0, 10, 20].each_with_index.map do |n, idx|
  n + idx
end
# => [0, 11, 22]
```

Excercise: Whack-A-Mole

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Make a whack-a-mole game! Randomly populate a 20-element array with true or false. Take 5 guesses from the user that correspond to array indices. Tell them how many moles (trues) they hit. Note: `my_str.split(" ")` splits a string into an array using `" "` as a separator. `my_ary.join(" ")` does the opposite.

```
Enter 5 guesses:
1, 4, 7, 12, 17
You got 3 right.
The moles were at 0, 1, 5, 7, 12, 19.
```

Solution: Whack-A-Mole

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
moles = Array.new(20) { rand(4) == 0 }
puts "Enter 5 guesses:"
guesses = gets.strip.split(" ").map{|s| s.to_i}

correct = guesses.select{|guess| moles[guess]}.length
puts "You got #{correct} right."

mole_locs = []
moles.each_with_index do |mole, idx|
  if mole
    mole_locs.push idx
  end
end
puts "The moles were at #{mole_locs.join(" ")}"
```

Slightly More Clever

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
mole_locs = moles.each_with_index.select do |pair|
  mole_idx_pair[0]
end.map do |pair|
  mole_idx_pair[1]
end
```

```ruby
mole_locs = moles.each_with_index.
                  select(&:first).
                  map(&:last)
```

Symbols

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Lightweight strings. Used instead of constants; as dictionary keys.

```
"a".object_id # => 16020460
"a".object_id # => 16214940
:a.object_id  # => 416808
:a.object_id  # => 416808
```

Hashes

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

A Hash (a.k.a. dictionary, HashMap, etc.) is a key-value store.

```
prices = {:cow => 1500, :pig => 800}
prices[:cow]           # => 1500
prices.include? :pig   # => true
prices.invert          # => {1500 => :cow, 800 => :pig}
```

Keys can be mutable, but if they change, you need to call `hsh.rehash`

Iterating over Hashes

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
rooms = {
    :ryan_putz      => "M213",
    :james_tetazoo => "Wa308",
    :jack_florey    => "B666"
}
rooms.each do |k, v|
    puts "Student: #{k}"
    puts "Room:    #{v}"
end

rooms.each do |pair|
    puts "Student: #{pair[0]}"
    puts "Room:    #{pair[1]}"
end
```

It's actually passing a 2-element array to the block, but Ruby unpacks it

Default Values

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
scores = Hash.new(0)
scores[:jack]   # => 0
scores[:james] # => 0

scores[:jack] += 1
scores[:jack]   # => 1
```

Tangent: Scan

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

`str.scan /regex/` returns all matches for the regular expression in the string. Very useful for pulling words out of text:

```
my_string = "Jack: he jumps over candlesticks!"
my_string.scan /\w+/
  #=> ["Jack", "he", "jumps", "over", "candlesticks"]
```

FYI, `str.split /some regex/` works too.

I/O

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
file_contents = IO.read("some/file.txt")
array_of_lines = IO.readlines("some/file.txt")

File.open("some/file.txt", "r") do |f|
  first_line = f.readline
end
File.open("some/file.txt", "w") do |f|
  f << "replaces content"
  f.puts "puts also works (and adds a newline)"
end
File.open("some/file.txt", "a") do |f|
  f << "appends content"
end
```

I/O Modes

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Mode	Meaning
`"r"`	Read-only, starts at beginning of file (default mode).
`"r+"`	Read-write, starts at beginning of file.
`"w"`	Write-only, truncates existing file to zero length or creates a new file for writing.
`"w+"`	Read-write, truncates existing file to zero length or creates a new file for reading and writing.
`"a"`	Write-only, starts at end of file if file exists, otherwise creates a new file for writing.
`"a+"`	Read-write, starts at end of file if file exists, otherwise creates a new file for reading and writing.

Exercise: Word Frequency

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Create a program that accepts a filename as an argument and prints out the file's word frequency. `ARGV` is an array of the program's arguments.

```
$ ruby word_count.rb some_file.txt
foo:  10
bar:  20
hello:  5
world:  5
```

Solution: Word Frequency

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
freqs = Hash.new(0)
IO.read(ARGV[0]).scan(/\w+/).each do |word|
  freqs[word] += 1
end

freqs.each do |word, freq|
  puts "#{word}: #{freq}"
end
```

Sorting

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
[3, 5, 1, 6].sort # => [1, 3, 5, 6]

["james", "jim", "jameson"].sort_by do |s|
  s.length
end
# => ["jim", "james", "jameson"]

# equivalently:
["james", "jim", "jameson"].sort_by &:length
```

Exercise: Sorted Word Frequency

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Make your word frequency counter print out sorted by frequency. `my_ary.to_a` converts a hash to an array of `[key, value]` pairs.

```
$ ruby word_count_better.rb some_file.txt
bar:  20
foo:  10
hello:  5
world:  5
```

Solution: Better Word Frequency

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
freqs = Hash.new(0)
IO.read(ARGV[0]).scan(/\w+/).each do |word|
  freqs[word] += 1
end

freqs.to_a.
      sort_by(&:last).
      reverse.
      each do |word, freq|

  puts "#{word}: #{freq}"
end
```

Defining Methods

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def sum_three(a, b, c)
  return a + b + c
end
sum_three(1, 2, 3) # => 6

def sum_three a, b, c # parens are optional
  return a + b + c
end
sum_three(1, 2, 3) # => 6

def sum_three a, b, c
  a + b + c # return is optional
end
sum_three(1, 2, 3) # => 6
```

Defining Methods With Flexible Arguments

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def increment n, amount=1
  n + amount
end

increment 1, 5 # => 6
increment 1    # => 2

def sum *nums
  sum = 0
  nums.each {|n| sum += n}
  sum
end

sum 1, 4, 10 # => 15
```

Defining Methods With Blocks

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def yield_multiples_of(a)
  yield a
  yield a*2
  yield a*3
end

yield_multiples_of 10 do |multiple|
  puts multiple
end
# prints 10, 20, 30
```

Yielding Multiple Values

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def times_ten(a, b, c)
  yield [a, a*10]
  yield [b, b*10]
  yield [c, c*10]
end

times_ten 1, 2, 3 do |normal, times_ten|
  puts "#{normal} times ten is #{times_ten}"
end
```

Defining Methods With Blocks: Alternative

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def multiples_of a, &b
  b.call a
  b.call a*2
  b.call a*3
end

multiples_of 10 do |multiple|
  puts multiple
end
# prints 10, 20, 30
```

Passing Around Procs

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def multiples_of a, &b
  b.call a
  b.call a*2
  b.call a*3
end

def multiples_of_ten &b
  multiples_of 10, &b
end

multiples_of_ten do |multiple|
  puts multiple
end
# prints 10, 20, 30
```

Open Objects

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Objects can have methods added at runtime.

```ruby
mit = Object.new

def mit.motto
  "IHTFP"
end

mit.motto # => "IHTFP"
```

Exercise: Word Frequency Method

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Make your word frequency counter into a method that takes a file name and yields word, frequency pairs.

```
frequencies("foo.txt") do |word, freq|
  puts "#{word} occurred #{freq} times"
end
```

Solution: Even Better Word Frequency

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def frequencies file, &block
  freqs = Hash.new(0)
  IO.read(file).scan(/\w+/).each do |word|
    freqs[word] += 1
  end

  freqs.each &block
end
```

Classes

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Sage
  def initialize
    @wisdom = []
  end

  def add_wisdom new_wisdom
    @wisdom.push new_wisdom
  end

  def wisdom
    @wisdom.sample # returns a random element
  end
end

sage = Sage.new
sage.add_wisdom "Are rhinos just fat unicorns?"
sage.add_wisdom "Cheer up, the worst is yet to come."
sage.add_wisdom "Rule 0: Everything is gonna be okay."

sage.wisdom # => "Rule 0: Everything is gonna be okay."
```

“Public” instance variables

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

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

  def name
    @name
  end

  def name= new_name
    @name = new_name
  end
end

stud = Student.new "Jack Florey"
stud.name # => "Jack Florey"
```

“Public” instance variables: The Easy Way

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Student
  attr_reader :name
  attr_writer :name

  def initialize name
    @name = name
  end
end

stud = Student.new "Jack Florey"
stud.name # => "Jack Florey"
stud.name = "James E. Tetazoo"
stud.name # => "James E. Tetazoo"
```

"Public" instance variables: The Even Easier Way

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Student
  attr_accessor :name

  def initialize name
    @name = name
  end
end

stud = Student.new "Jack Florey"
stud.name # => "Jack Florey"
stud.name = "James E. Tetazoo"
stud.name # => "James E. Tetazoo"
```

"Public" instance variables: Using Structs

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
Student = Struct.new :first_name, :last_name
stud = Student.new "Jack", "Florey"
stud.first_name  # => "Jack"
stud[:last_name] # => "Florey"
```

Class Methods

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Student
  attr_accessor :first_name, :last_name

  def initialize first_name, last_name
    @first_name = first_name
    @last_name = last_name
  end
end

def Student.from_full_name name
  parts = name.scan /\w+/
  Student.new parts.first, parts.last
end

stud = Student.from_full_name "James E. Tetazoo"
stud.first_name # => "James"
stud.last_name  # => "Tetazoo"
```

Class Methods: The Better Way

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Student
  attr_accessor :first_name, :last_name

  def initialize first_name, last_name
    @first_name = first_name
    @last_name = last_name
  end

  def self.from_full_name name
    parts = name.scan /\w+/
    Student.new parts.first, parts.last
  end
end

stud = Student.from_full_name "James E. Tetazoo"
stud.first_name # => "James"
stud.last_name  # => "Tetazoo"
```

Class Variables

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Door
  @@master_key = nil

  def initialize key
    @key = key
  end

  def unlock key
    key == @key or key == @@master_key
  end

  def self.set_master_key key
    @@master_key = key
  end
end

Door.set_master_key "open_sesame"
d1 = Door.new "1234"
d1.unlock "master_key" # => true
```

Referencing other Methods

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Multiplier
  def double n
    n*2
  end

  def quadruple n
    double(double(n))
  end

  # equivalently:
  def quadruple n
    self.double(self.double(n))
  end
end

Multiplier.new.quadruple 10 # => 40
```

Referencing other Methods

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Multiplier
  def double n
    n*2
  end

  def quadruple n
    double(double(n))
  end

  # equivalently:
  def quadruple n
    self.double(self.double(n))
  end
end

Multiplier.new.quadruple 10 # => 40
```

Excercise: Social Network

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

We're going to build a `Person` class for use in a social network. It will keep track of its friends, and the class will let you look up people by name.

```
g = Person.new "George"
a = Person.new "Arthur", g
f = Person.new "Fred", a, g
f.friends # => [a, g]
a.friends # => [g]
Person.find "Arthur" # => a
```

BONUS: Implement a method to find the shortest path between people in the graph.

```
a = Person.new "A"
b = Person.new "B", a
c = Person.new "C"
d = Person.new "D", b, c
d.distance_to a # => 2
                # D -> B -> A = 2
a.distance_to b # => nil
```

Solution: Social Network


```ruby
class Person
  attr_accessor :friends
  @@registry = {}

  def initialize name, *friends
    @name = name
    Person.register name, self
    @friends = friends
  end

  def self.register name, person
    @@registry[name] = person
  end

  def self.find name
    @@registry[name]
  end
end
```

Solution: Social Network BONUS

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Person
  def distance_to person
    level = 0
    all_found = [self]
    last_level = [self]
    loop do
      return level if all_found.include? person

      this_level = last_level.map(&:friends).flatten.uniq

      last_size = all_found.length
      all_found = (all_found + this_level).uniq
      return nil if  all_found.length == last_size

      level += 1
      last_level = this_level
    end
  end
end
```

Inheritance

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Formatter
  def format str
    str
  end
end

class BoldFormatter < Formatter
  def format str
    "**" + super + "**"
  end
end

class CapitalBoldFormatter < BoldFormatter
  def format str
    super str.upcase
  end
end

CapitalBoldFormatter.new.format "hello"
  # => "**HELLO**"
```

Inheritance: Using Structs

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Greeter < Struct.new(:name)
  def greet
    "Hi! My name is #{self.name}!"
  end
end

Greeter.new("Peter").greet
  # => "Hi! My name is Peter!"
```

Exercise: Animals

Create an `Animal` class that takes a name and a sound. Then make specialized subclasses that only need a name.

```
a = Animal.new("Ben Bitdiddle", "blah")
a.speak # => "Ben Bitdiddle says blah"
c = Cow.new("Bessie")
c.speak # => "Bessie saws mooooooo"
h = Hacker.new("Alyssa P.")
h.speak # => "Alyssa P. says 10010101"
```

Solution: Animals

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

```ruby
class Animal
  def initialize name, sound
    @name, @sound = name, sound
  end
  def speak
    "#{@name} says #{@sound}"
  end
end

class Cow < Animal
  def initialize name
    super name, "mooooooo"
  end
end

class Hacker < Animal
  def initialize name
    super name, "10010010"
  end
end
```

Public/Private/Protected

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Public	Callable from anywhere
Protected	Callable from any instance of the defining class or any of its subclasses
Private	Cannot have an explicit receiver (even `self`).

Public/Private/Protected: Syntax

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Foo
  def pub_method; end

  private
  def priv_method; end
  def other_private; end

  public
  def other_pub; end

  protected
  def prot_method; end
end
```

Public/Private/Protected: Alternate Syntax

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Foo
  def pub_method; end

  def priv_method; end
  def other_private; end
  private :priv_method, :other_private

  def other_pub; end

  def prot_method; end
  protected :prot_method
end
```

Private vs Protected: Outside

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
f = Foo.new
f.priv_method # => ERROR
f.prot_method # => ERROR
```

Private vs Protected: Inside

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Foo
  def bar
    self.prot_method # => works
    self.priv_method # => ERROR
    priv_method      # => works
  end
end
```

Private vs Protected: Same Class

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Foo
  def bar other_foo
    other_foo.prot_method # => works
    other_foo.priv_method # => ERROR
  end
end
```

Private vs Protected: Subclass

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
class SubFoo < Foo
  def bar
    prot_method       # => works
    priv_method       # => works
  end
end
```

Modules for Grouping

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Module Use Case 1: Grouping

```ruby
module Encryptor
  def self.gen_key
    rand(255)
  end
  def self.encrypt str, key
    str.bytes
       .map{|byte| sprintf "%02x", (byte ^ key)}
       .join
  end
  def self.decrypt str, key
    str.scan(/../)
       .map{|byte| (byte.to_i(16) ^ key).chr}
       .join
  end
end
key = Encryptor.gen_key                 # => 171
Encryptor.encrypt "hello", key          # => "c3cec7c7c4"
Encryptor.decrypt "c3cec7c7c4", key # => "hello"
```

Modules as Namespaces

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Module Use Case 2: Namespaces

```ruby
module MIT
  LOCATION = "Cambridge, MA"

  class Person
    # ...
  end

  class Student < Person
    # ...
  end
end

MIT::LOCATION # => "Cambridge, MA"
MIT::Student.new
```

Module as Mixins

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics

Inheritance

Method Visibility

Modules

Extension

Meta-programming

Wrap-Up

Module Use Case 3: Mixins

```ruby
module Doubler
  def double
    self.single * 2
  end
end

class Person < Struct.new(:age)
  include Doubler

  def single
    self.age
  end
end

Person.new(5).double # => 10
```

Enumerable

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Mixin `Enumerable` to any class with a `each` method to get lots of iterators:

```
class FirstThree
  include Enumerable
  def each
    yield 1
    yield 2
    yield 3
  end
end

FirstThree.new.map{|x| x * 2} # => [2, 4, 6]
```

Mixins

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

What are mixins good for?

- Multiple inheritance
- Common interfaces to similar classes

Exercise: Pettable

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Create a `Pettable` mixin. It gets mixed into classes that have a `name` method:

```
class Bunny
  include Pettable
  def name
    "Fluffy"
  end
end
Bunny.new.pet # => "Awwww... Fluffy likes it!"
```

Solution: Pettable

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
module Pettable
  def pet
    "Awww... #{self.name} likes it!"
  end
end
```

Numeric Classes

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

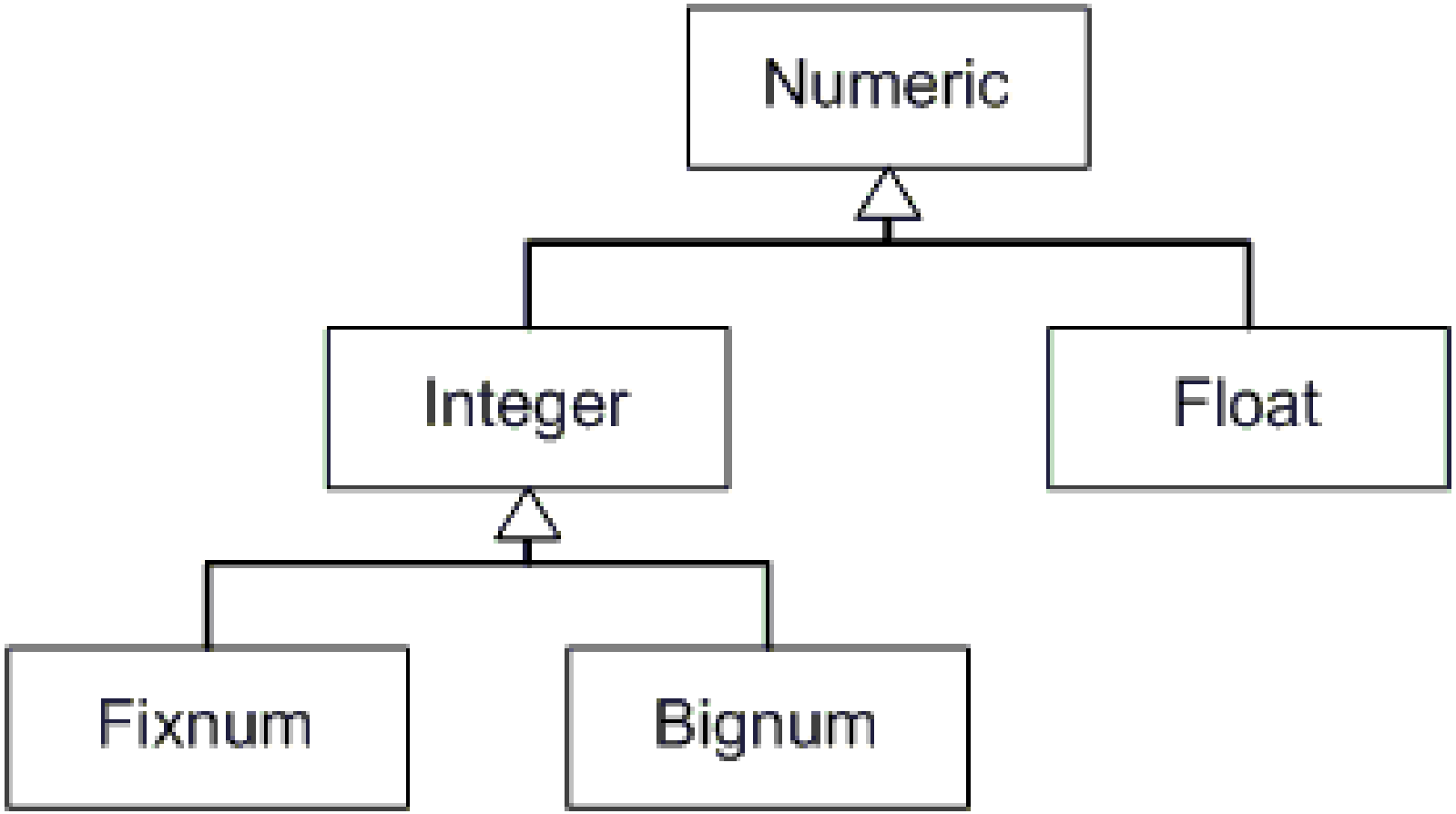
Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up



Extending Classes

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Numeric
  def cubed
    self ** 3
  end
end

4.cubed # => 64
```

Exercise: Roman Numerals

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Create a module to convert to/from roman numerals. Extend String and Integer to support the conversions.

```
Roman.from_roman("XIV")  # => 14
Roman.to_roman(49)       # => "XLIX"
"XIV".from_roman         # => 14
49.to_roman              # => 49
"INVALID".from_roman     # => nil
```

Exercise: Roman Numerals: Hint 1

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Hint 1: inside a block, you can use `redo` to re-start execution of the block. Here's an example that forces the random-number generator to generate six sixes:

```
6.times do |i|
  puts i
  if rand(7) != 6
    redo
  end
end
# might print 1, 1, 1, 1, 2, 2, 3, 3, 3...
```

Exercise: Roman Numerals: Hint 2

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Hint 2: You can also use `break` inside a block to cancel the method. Here's one that will break if the RNG ever generates a 6:

```
6.times do
  if rand(7) != 6
    break
  end
end
puts done
# might print 1, 2, 3
```

Exercise: Roman Numerals: Hint 3

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

To get you started:

```
ROMAN = [
  ['M',  1000],
  ['CM', 900],
  ['D',  500],
  ['CD', 400],
  ['C',  100],
  ['XC', 90],
  ['L',  50],
  ['XL', 40],
  ['X',  10],
  ['IX', 9],
  ['V',  5],
  ['IV', 4],
  ['I',  1]
]
```

Solution: Roman Numerals: Part 1

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
module Roman
  ROMAN = [...]
  def self.to_roman i
    # ...
  end
  def self.from_roman str
    # ...
  end
end

class String
  def from_roman
    Roman.from_roman self
  end
end
class Fixnum
  def to_roman
    Roman.to_roman self
  end
end
```

Solution: Roman Numerals: Part 2

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def self.to_roman i
  r = ''
  ROMAN.each do |sym, val|
    if i >= val
      i -= val
      r += sym
      redo
    end
  end
  r
end
```

Solution: Roman Numerals: Part 3

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
def self.from_roman str
  r = 0
  while str.length > 0
    found = false
    ROMAN.each do |sym, val|
      if str.start_with? sym
        str = str[(sym.length)..-1]
        r += val
        found = true
        break
      end
    end

    unless found
      return nil
    end
  end
  r
end
```

Sending Messages

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Method invocation is a lie. It's messages all the way down

```ruby
"string".send(:reverse) # => "gnirts"
```

Oh, and private methods are a lie:

```ruby
class Secrets
  private
  def secret
    "my private key"
  end
end
s = Secrets.new
s.secret        # => error
s.send :secret # => "my private key"
```

Accessing Constants

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Constants can be accesses in the same way:

```ruby
module ImportantNumbers
  PI = 3.14159
end
ImportantNumbers.const_get :PI # => 3.14159
```

Method Missing

When you call a method that doesn't exist, Ruby invokes `method_missing`. By default, this throws an error, but we can make it do other cool stuff:

```
class Parrot
  def method_missing method, *args, &block
    "Squak! #{method}! #{method}"
  end
end
p = Parrot.new
p.treasure # => "Squak! treasure! treasure!"
```

Const Missing

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Same with constants:

```ruby
class Hello
  def self.const_missing name
    "WTF?"
  end
end
Hello::WORLD # => "WTF?"
```

Example: VCR

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class VCR
  def initialize
    @records = []
  end
  def method_missing method, *args, &block
    @records.push [method, args, block]
  end
  def replay obj
    @records.each do |method, args, block|
      obj = obj.send method, *args, &block
    end
    obj
  end
end

vcr = VCR.new
vcr.upcase
vcr.reverse
vcr + " world"
vcr.replay "hello" # => "OLLEH world"
```

Eigenclasses

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

It's what we're accessing when we define a method for a specific object

```
a = Object
class << a
  def b
    10
  end
end
a.b # => 10
```

Eigenclasses for Class Methods

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Stuff inside class blocks is evaluated in the context of the class, so...

```
class Greeter
  class << self
    def greet name
      "Hey there, #{name}!"
    end
  end
end
Greeter.greet "y'all" # => "Hey there, y'all!"
```

Alias Method

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

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

  def name
    @name
  end

  alias_method :moniker, :name
end

Person.new("Ben").moniker # => "Ben"
```

Wrapping Methods

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```ruby
class Fixnum
  alias_method :old_times, :*
  def * other
    if other > 1_000_000
      "Too big!"
    else
      old_times(other)
    end
  end
end
10 * 10 # => 100
10 * 10_000_000 # => "Too big!"
```

The Global Context

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

```
self        # => main
self.class  # => Object
```

`main` is just and instance of `Object`: we can defined global methods by defining them on `Object`

Down the Rabbit Hole

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Roman numeral literals:

```
class Object
  class << self
    alias_method :const_missing_old, :const_missing
    def const_missing c
      i = RomanNumerals.from_roman(c.to_s)
      if i
        return i
      else
        return const_missing_old(c)
      end
    end
  end
end
XIV         # => 14
IV + XIX    # => 23
```

But Why?

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

"Unix was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things."
– Doug Gwyn

We're Done!

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

Questions?

Eval

Caffeinated Crash Course in Ruby

Ben Weissmann

Introduction

The Basics

Arrays & Iterators

Hashes

I/O

Methods

Classes

The Basics
Inheritance
Method Visibility
Modules
Extension

Meta-programming

Wrap-Up

http://tinyurl.com/ml-ruby