10 Things You Didn’t Know Ruby Could Do
James Edward Gray II

- I’m a regular on the Ruby Rogues podcast
- I’ve written a lot of documentation and code for Ruby
- This is my first time in Hawai’i
10 Things You Didn’t Know Rails Could Do

RailsConf 2012
42

10 Things You Didn’t Know Rails Could Do

RailsConf 2012
10 Things You Didn’t Know Ruby Could Do
10 Things You Didn’t Know Ruby Could Do
101 Things You Didn’t Know Ruby Could Do
MRI Ruby 1.9.3-p194...
Compiler Tricks
Dear Rubyists:

Did you know that Ruby can even read your emails?

#!/usr/bin/env ruby -w

puts "It's true."

__END__

I told you it could.

James Edward Gray II

Ruby Can Read Your Email

Or any other text content with Ruby code in it
Dear Rubyists:

Did you know that Ruby can even read your emails?

#!/usr/bin/env ruby -w
puts "It's true."

__END__

I told you it could.

James Edward Gray II

$ ruby -x email.txt
It's true.
Dear Rubyists:

Did you know that Ruby can even read your emails?

#!/usr/bin/env ruby -w
puts "It's true."

__END__

I told you it could.

James Edward Gray II

$ ruby -x email.txt
It's true.
Dear Rubyists:

Did you know that Ruby can even read your emails?

```ruby
#!/usr/bin/env ruby -w
puts "It's true."
__END__
I told you it could.
James Edward Gray II
```

$ ruby -x email.txt
It's true.
Dear Rubyists:

Did you know that Ruby can even read your emails?

#!/usr/bin/env ruby -w

puts "It's true."

__END__

I told you it could.

James Edward Gray II

$ ruby -x email.txt
It's true.
Storing Data in Your Code
Or is it DATA?

#2 From David Brady
Storing Data in Your Code

Or is it DATA?

#2 From David Brady
Storing Data in Your Code

Or is it DATA?

#2 From David Brady
Storing Data in Your Code

Or is it DATA?

#2 From David Brady
A Cheat of a Quine

Most quine rules disallow using IO

#3 From David Brady
A Cheat of a Quine

Most quine rules disallow using IO

#3 From David Brady
A Cheat of a Quine

Most quine rules disallow using IO

#3 From David Brady
There Can Be Only One

Use a tricky lock to make your script exclusive

#4 From Ara T. Howard
There Can Be Only One

Use a tricky lock to make your script exclusive

#4 From Ara T. Howard
There Can Be Only One

Use a tricky lock to make your script exclusive
There Can Be Only One

Use a tricky lock to make your script exclusive

#4 From Ara T. Howard
pos = DATA.pos
list = DATA.readlines

if ARGV.empty?
    puts list.shift
else
    list.push(*ARGV)
end

DATA.reopen(__FILE__, "r+")
DATA.truncate(pos)
DATA.seek(pos)
DATA.puts list

__END__
Service-Oriented Design with Ruby and Rails
Practical Object-Oriented Design in Ruby

Your Source File, The Database
A dirty trick to carry some data with the code
Your Source File, The Database

A dirty trick to carry some data with the code

#5
pos = DATA.pos
list = DATA.readlines

if ARGV.empty?
    puts list.shift
else
    list.push(*ARGV)
end

DATA.reopen(__FILE__, "r+")
DATA.truncate(pos)
DATA.seek(pos)
DATA.puts list

__END__
Service-Oriented Design with Ruby and Rails
Practical Object-Oriented Design in Ruby

$ ruby reading_list.rb
   'Service-Oriented Design with Ruby and Rails'
   'Practical Object-Oriented Design in Ruby'
$ ruby reading_list.rb
Service-Oriented Design with Ruby and Rails
$ ruby reading_list.rb
Practical Object-Oriented Design in Ruby

Your Source File, The Database
A dirty trick to carry some data with the code
See How Ruby Reads Your Code

All you have to do is ask

---

#6 From Eric Hodel
See How Ruby Reads Your Code

All you have to do is ask

#6 From Eric Hodel
See How Ruby Reads Your Code

All you have to do is ask

#6 From Eric Hodel
$ ruby -e 'puts { is_this_a_block }' --dump parsetree_with_comment

Available With Comments

Ruby can even explain it to you

#7 From Eric Hodel
Available With Comments

Ruby can even explain it to you

#7 From Eric Hodel
Ruby can even explain it to you

Available With Comments

Ruby can even explain it to you
$ ruby -e 'ft = 40 + 2; p ft' --dump insns
== disasm: <RubyVM::InstructionSequence:<main>@-e>======================
local table (size: 2, argc: 0 [opts: 0, rest: -1, post: 0, block: -1] s1)
[ 2] ft
0000 trace            1                                               (   1)
0002 putobject        40
0004 putobject        2
0006 opt_plus         <ic:2>
0008 setdynamic       ft, 0
0011 trace            1
0013 putsself
0014 getdynamic       ft, 0
0017 send             :p, 1, nil, 8, <ic:1>
0023 leave
$ ruby -e 'ft = 40 + 2; p ft' --dump insns
== disasm: <RubyVM::InstructionSequence, <main>@-e>======================
local table (size: 2, argc: 0 [opts: 0, rest: -1, post: 0, block: -1] s1)
[ 2] ft
0000 trace            1                                               (   1)
0002 putobject        40
0004 putobject        2
0006 opt_plus         <ic:2>
0008 setdynamic       ft, 0
0011 trace            1
0013 putsself
0014 getdynamic       ft, 0
0017 send             :p, 1, nil, 8, <ic:1>
0023 leave
Watch Ruby’s Parser Work

See how parse.y thinks

#9 From Eric Hodel
$ ruby -e 'arg = ARGV.shift or abort "No arg"' --dump yydebug
Starting parse
Entering state 0
Reducing stack by rule 1 (line 782):
-> $$ = nterm $$@1 ()
Stack now 0
Entering state 2
Reading a token: Next token is token tIDENTIFIER ()
Shifting token tIDENTIFIER ()
Entering state 35
Reading a token: Next token is token '=' ()
Reducing stack by rule 474 (line 4275):
  $1 = token tIDENTIFIER ()
-> $$ = nterm user_variable ()
Stack now 0 2
...

Watch Ruby’s Parser Work
See how parse.y thinks

#9 From Eric Hodel
Inspecting the Source

For those of us who belong to “The Ridiculous Church of 80-Character Lines”

#10 From Eric Hodel
Inspecting the Source

For those of us who belong to “The Ridiculous Church of 80-Character Lines”

#10 From Eric Hodel

```ruby
SCRIPT_LINES__ = {}

require_relative "better_be_well_formed_code"

# format: {"file_name.rb" => ["line 1", "line 2", ...]}
if SCRIPT_LINES__.values.flatten.any? { |line| line.size > 80 }
  abort "Clean up your code first!"
end
```
Inspecting the Source

For those of us who belong to “The Ridiculous Church of 80-Character Lines”

#10 From Eric Hodel
SCRIPT_LINES__ = { }

require_relative "better_be_well_formed_code"

# format:  {"file_name.rb" => ["line 1", "line 2", ...]}
if SCRIPT_LINES__.values.flatten.any? { |line| line.size > 80 }
  abort "Clean up your code first!"
end

Inspecting the Source
For those of us who belong to “The Ridiculous Church of 80-Character Lines”

#10 From Eric Hodel
def factorial(n, result = 1)
    if n == 1
        result
    else
        factorial(n - 1, n * result)
    end
end

p factorial(30_000)
def factorial(n, result = 1)
    if n == 1
        result
    else
        factorial(n - 1, n * result)
    end
end

p factorial(30_000)

../../factorial.rb:2: stack level too deep (SystemStackError)

Tail Call Optimization

Yeah, it’s in there

#11 From Aaron Patterson
RubyVM::InstructionSequence.compile_option = { tailcall_optimization: true, trace_instruction: false }

eval <<end
  def factorial(n, result = 1)
    if n == 1
      result
    else
      factorial(n - 1, n * result)
    end
  end
end

p factorial(30_000)

---

Tail Call Optimization

Yeah, it’s in there

#11 From Aaron Patterson
RubyVM::InstructionSequence.compile_option = { tailcall_optimization: true, trace_instruction: false }

eval <<end
  def factorial(n, result = 1)
    if n == 1
      result
    else
      factorial(n - 1, n * result)
    end
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

eval <<end
  def factorial(n, result = 1)
    if n == 1
      result
    else
      factorial(n - 1, n * result)
    end
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

def factorial(n, result = 1)
  if n == 1
    result
  else
    factorial(n - 1, n * result)
  end
end

# 11 From Aaron Patterson

Tail Call Optimization

Yeah, it’s in there
RubyVM::InstructionSequence.compile_option = { tailcall_optimization: true, 
  trace_instruction: false }
eval <<end
  def factorial(n, result = 1)
    if n == 1
      result
    else
      factorial(n - 1, n * result)
    end
  end
end
p factorial(30_000)

Tail Call Optimization
Yeah, it’s in there

#11 From Aaron Patterson
eval <<end
  def factorial(n, result = 1)
    if n == 1
      result
    else
      factorial(n - 1, n * result)
    end
  end
end

p factorial(30_000)

RubyVM::InstructionSequence.compile_option = {
  tailcall_optimization: true,
  trace_instruction: false
}

Tail Call Optimization

Yeah, it’s in there

#11 From Aaron Patterson
Syntax
Pass Some Arguments Up

Helps add functionality to methods
Pass Some Arguments Up

Helps add functionality to methods
Pass Some Arguments Up

Helps add functionality to methods
Pass The Same Arguments Up

Remember, super is a magical keyword
class Parent
  def show_args(*args, &block)
    p [*args, block]
  end
end

class Child < Parent
  def show_args(a, b, c)
    super
  end
end

Child.new.show_args(:a, :b, :c) { :block }
class Parent
    def show_args(*args, &block)
        p [*args, block]
    end
end

class Child < Parent
    def show_args(a, b, c)
        super
    end
end

Child.new.show_args(:a, :b, :c) { :block }

[:a, :b, :c, #<Proc:0x007fed2c887568@.../pass_the_same_arguments_up.rb:13>]

---

Pass The Same Arguments Up

Remember, super is a magical keyword

#13 From Daniel Azuma
class Parent
  def show_args(*args)
    p args
  end
end

class Child < Parent
  def show_args(a, b, c)
    a.upcase!  # not too surprising
    b = "Wow!"  # very surprising
    super
  end
end

Child.new.show_args("a", "b", "c")
Pass Modified Arguments Up

Scary magic

#14 From Daniel Azuma
class Parent
  def show_args(*args)
    p args
  end
end

class Child < Parent
  def show_args(a, b, c)
    a.upcase!    # not too surprising
    b = "Wow!"   # very surprising
    super
  end
end

Child.new.show_args("a", "b", "c")
class Parent
  def show_args(*args)
    p args
  end
end

class Child < Parent
  def show_args(a, b, c)
    a.upcase! # not too surprising
    b = "Wow!" # very surprising
    super
  end
end

Child.new.show_args("a", "b", "c")

["A", "Wow!", "c"]

Pass Modified Arguments Up
Scary magic
class Parent
  def show_args(*args)
    p args
  end
end

class Child < Parent
  def show_args(a, b, c)
    super()
  end
end

Child.new.show_args(:a, :b, :c)

Pass No Arguments Up

The parentheses are required

#15 From Daniel Azuma
Pass No Arguments Up

The parentheses are required
Pass No Arguments Up

The parentheses are required

#15 From Daniel Azuma
Pass No Block Up

How to make a block disappear

#16 From Marten Veldthuis
Pass No Block Up

How to make a block disappear
Pass No Block Up

How to make a block disappear

```ruby
class Parent
  def show_block(&block)
    p block
  end
end
class Child < Parent
  def show_block
    super(&nil)
  end
end
Child.new.show_block { :block }
```

nil
Ruby will tell you if a parent method is available to delegate to.
Asking If You Can Pass Up

Ruby will tell you if a parent method is available to delegate to

#17 From Avdi Grimm
Asking If You Can Pass Up

Ruby will tell you if a parent method is available to delegate to

```ruby
class DontDelegateToMe; end
class DelegateToMe; def delegate; "DelegateToMe" end end
module DelegateIfICan
def delegate
  if defined? super
    "Modified: #{super}"
  else
    "DelegateIfICan"
  end
end
end
puts DelegateToMe.new.extend(DelegateIfICan).delegate
puts DontDelegateToMe.new.extend(DelegateIfICan).delegate
```

#17 From Avdi Grimm
Lambda Literals

Also known as the “stabby lambda”
Lambda Literals

Also known as the “stabby lambda”
Lambda Literals

Also known as the “stabby lambda”

minimal = -> { p :called }
minimal.call

loaded = ->(arg, default = :default, &block) { p [arg, default, block] }
loaded.call(:arg) { :block }
Lambda Literals

Also known as the “stabby lambda”
Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us

#19 From Benjamin Fleischer
Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us

#19 From Benjamin Fleischer
Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us

#19 From Benjamin Fleischer
Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us
var = :var
object = Object.new

object.define_singleton_method(:show_var_and_block) do |
  &block|
  p [var, block]
end

object.show_var_and_block { :block }
Blocks Can Now Take Blocks

This trick helps with metaprogramming
Blocks Can Now Take Blocks

This trick helps with metaprogramming

```ruby
var = :var
object = Object.new

object.define_singleton_method(:show_var_and_block) do |&block|
  p [var, block]
end

object.show_var_and_block { :block }
```

[[:var, #<Proc:0x007fef59908f30@/.../blocks_can_now_take_blocks.rb:8>]]
class Callable
  def call
    :my_own_class
  end
end

def call()
  :lambda
end

p -> { :lambda }.()
p [ ].method(:class).()
p Callable.new.()
The New Call Anything Syntax

Or should we call it the “Call Me Maybe” syntax?
class Callable
    def call
        :my_own_class
    end
end

p -> { :lambda }.()
p[].method(:class).()
p Callable.new.()

The New Call Anything Syntax

Or should we call it the “Call Me Maybe” syntax?
Symbol#to_proc Takes Arguments

This probably isn’t a great thing to abuse, but still… shiny

#22 From David Brady
This probably isn’t a great thing to abuse, but still… shiny

```ruby
to_s_proc = :to_s.to_proc  # or: lambda(&:to_s)
receiver = 255
arg = 16
puts to_s_proc[receiver, arg]
```
Symbol\#to_proc Takes Arguments

This probably isn’t a great thing to abuse, but still… shiny
inject() a Symbol

This is even shorter than using Symbol#to_proc
`inject()` a Symbol

This is even shorter than using `Symbol#to_proc`
injected a Symbol

This is even shorter than using Symbol#to_proc

\[ p (1..10).inject(:*) \]  # instead of: injected(&:* )

3628800

#23
Case on Ranges

Case statements work with anything that defines ===, like Range

```ruby
age = rand(1..100)
p age

case age
  when -(Float::INFINITY..20
    puts "You're too young."
  when 21..64
    puts "You are the right age."
  when 65..Float::INFINITY
    puts "You're too old."
end
```
Case on Ranges

Case statements work with anything that defines `===`, like Range

```ruby
age = rand(1..100)
p age

case age
  when -Float::INFINITY..20
    puts "You're too young."
  when 21..64
    puts "You are the right age."
  when 65..Float::INFINITY
    puts "You're too old."
end
```
Case on Ranges

Case statements work with anything that defines ===, like Range

```ruby
age = rand(1..100)
p age

case age
when -Float::INFINITY..20
  puts "You're too young."
when 21..64
  puts "You are the right age."
when 65..Float::INFINITY
  puts "You're too old."
end

96
You're too old.
```
Case on Date Ranges

Range objects work in a case and Date objects work as a Range endpoint

```ruby
require "date"

start_of_aloha_ruby_conf = Date.new(2012, 10, 8)
end_of_aloha_ruby_conf = Date.new(2012, 10, 9)

case Date.today
  when Date.new...start_of_aloha_ruby_conf
    puts "Anticipation is building."
  when start_of_aloha_ruby_conf..end_of_aloha_ruby_conf
    puts "Mind being blown."
  when (end_of_aloha_ruby_conf + 1)..Date::Infinity
    puts "You've learned some Ruby while in paradise."
end
```
Case on Date Ranges

Range objects work in a case and Date objects work as a Range endpoint

#25 From Piotr Szotkowski
require "date"

start_of_aloha_ruby_conf = Date.new(2012, 10, 8)
end_of_aloha_ruby_conf = Date.new(2012, 10, 9)

case Date.today
  when Date.new...start_of_aloha_ruby_conf
    puts "Anticipation is building."
  when start_of_aloha_ruby_conf..end_of_aloha_ruby_conf
    puts "Mind being blown."
  when (end_of_aloha_ruby_conf + 1)..Date::Infinity
    puts "You've learned some Ruby while in paradise."
end

Mind being blown.

Case on Date Ranges

Range objects work in a case and Date objects work as a Range endpoint

#25 From Piotr Szotkowski
require "prime"

n = rand(1..10)
p n

case n
when lambda(&:prime?)
  puts "This number is prime."
when lambda(&:even?)
  puts "This number is even."
else
  puts "This number is odd."
end

Case on Lambdas
As of Ruby 1.9, lambdas (Proc objects) also define ===

#26 From Piotr Szotkowski
```ruby
require "prime"

n = rand(1..10)
p n

case n
when lambda(&:prime?)  puts "This number is prime."
when lambda(&:even?)   puts "This number is even."
else                     puts "This number is odd."
end
```

Case on Lambdas

As of Ruby 1.9, lambdas (Proc objects) also define ===

#26 From Piotr Szotkowski
require "prime"

n = rand(1..10)
p n

case n
when lambda(&:prime?)
  puts "This number is prime."
when lambda(&:even?)
  puts "This number is even."
else
  puts "This number is odd."
end

9
This number is odd.

Case on Lambdas
As of Ruby 1.9, lambdas (Proc objects) also define ===
def debug(name, content)
    puts "\%s: \%p" % [name, content]
end

d debug "Num", 42
d debug "Objects", { "Grays" => %w[James Dana Summer]}

A Formatted Output Syntax

Like sprintf(), but even shorter

#27 From Benjamin Fleischer
def debug(name, content)
    puts "\%s: \%p" % [name, content]
end

debug "Num", 42
debug "Objects", {"Grays" => %w[James Dana Summer]}
def debug(name, content)
    puts "\%s: \%p" % [name, content]
end

depbug "Num", 42
depbug "Objects", {"Grays" => %w[James Dana Summer]}
def debug(name, content)
    puts "%s: %p" % [name, content]
end

dump "Num", 42
dump "Objects", {"Grays" => %w[James Dana Summer]}

Num: 42
Objects: {"Grays"=>["James", "Dana", "Summer"]}

A Formatted Output Syntax

Like sprintf(), but even shorter

#27 From Benjamin Fleischer
order = {"Item 1" => 10, "Item 2" => 19.99, "Item 3" => 4.50}

item_size  = (['Item'] + order.keys).map(&:size).max
price_size = (['Price'].size +
    order.values.map { |price| "$%.2f" % price ).size ).max

puts "%<item>-#{item_size}s | %<price>#{price_size}s" %
    {item: "Item", price: "Price"}
puts "-" * (item_size + price_size + 3)
order.each do |item, price|
    puts "%<item>-#{item_size}s | $%<price>#{price_size - 1}.2f" %
        {item: item, price: price}
end

---

# Or By Name

A Ruby 1.9 enhancement to these format patterns
order = {"Item 1" => 10, "Item 2" => 19.99, "Item 3" => 4.50}

item_size = (["Item"] + order.keys).map(&:size).max
price_size = ( ["Price".size] + order.values.map { |price| ("$%.2f" % price).size } ).max

puts "%<item>-#{item_size}s | %<price>#{price_size}s" %
    {item: "Item", price: "Price"}
puts "-" * (item_size + price_size + 3)
order.each do |item, price|
    puts "%<item>-#{item_size}s | $%<price>#{price_size - 1}.2f" %
        {item: item, price: price}
end

Or By Name

A Ruby 1.9 enhancement to these format patterns
order = { "Item 1" => 10, "Item 2" => 19.99, "Item 3" => 4.50 }

item_size = (["Item"] + order.keys).map(&:size).max
price_size = ( ["Price"].size + 
            order.values.map { |price| ("$%.2f" % price).size }).max

puts "%<item>-#{item_size}s | %<price>#{price_size}s" % 
    { item: "Item", price: "Price"}
puts "-" * (item_size + price_size + 3)
order.each do |item, price|
    puts "%<item>-#{item_size}s | $%<price>#{price_size - 1}.2f" % 
        { item: item, price: price }
end

Or By Name

A Ruby 1.9 enhancement to these format patterns
Or By Name

A Ruby 1.9 enhancement to these format patterns
order = {"Item 1" => 10, "Item 2" => 19.99, "Item 3" => 4.50}

item_size = (["Item"] + order.keys).map(&:size).max
price_size = (["Price"].size + 
  order.values.map { |price| ("$%.2f" % price).size } ).max

puts "%<item>-#{item_size}s | %<price>#{price_size}s" % 
  {item: "Item", price: "Price"}
puts "-" * (item_size + price_size + 3)
order.each do |item, price|
  puts "%<item>-#{item_size}s | $%<price>#{price_size - 1}.2f" % 
    {item: item, price: price}
end

Or By Name

A Ruby 1.9 enhancement to these format patterns
order = {"Item 1" => 10, "Item 2" => 19.99, "Item 3" => 4.50}

item_size = (["Item"] + order.keys).map(&:size).max
price_size = ( ["Price"].size +
    order.values.map { |price| ("$%.2f" % price).size } ).max

puts "%<item>-#{item_size}s | %<price>#{price_size}s" %
    {item: "Item", price: "Price"}
puts "-" * (item_size + price_size + 3)
order.each do |item, price|
    puts "%<item>-#{item_size}s | $%<price>#{price_size - 1}.2f" %
        {item: item, price: price}
end

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item 1</td>
<td>$10.00</td>
</tr>
<tr>
<td>Item 2</td>
<td>$19.99</td>
</tr>
<tr>
<td>Item 3</td>
<td>$ 4.50</td>
</tr>
</tbody>
</table>
def create_post(title, summary, body)
    # ...
end

create_post("Aloha RubyConf", <<END_SUMMARY, <<END_BODY)
A multiline summary.
END_SUMMARY
And a multiline body.
END_BODY

Multiple HEREDOC’s
Ruby is smart enough to pick several off the same line
def create_post (title, summary, body)
    # ...
end

create_post("Aloha RubyConf", <<END_SUMMARY, <<END_BODY)
A multiline summary.
END_SUMMARY
And a multiline body.
END_BODY

Multiple HEREDOC’s
Ruby is smart enough to pick several off the same line
def create_post(title, summary, body)
    # ...
end

create_post("Aloha RubyConf", <<END_SUMMARY, <<END_BODY)
A multiline
summary.
END_SUMMARY
And a
multiline
body.
END_BODY

Multiple HEREDOC’s

Ruby is smart enough to pick several off the same line
Dodging a Warning

The “or equals” operator includes a defined? check
Dodging a Warning
The “or equals” operator includes a defined? check
Dodging a Warning

The “or equals” operator includes a defined? check
Dodging a Warning

The “or equals” operator includes a defined? check

```ruby
$VERBOSE = true

class WarnMe
  def var
    @var ||= 42
  end
end

p WarnMe.new.var
```

```
/.../dodging_a_warning.rb:5: warning: instance variable @var not initialized
42
```
Dodging a Warning

The “or equals” operator includes a defined? check

```
$VERBOSE = true

class WarnMe
  def var
    @var ||= 42
  end
end

p WarnMe.new.var
```
Dodging a Warning

The “or equals” operator includes a defined? check
Dodging a Warning

The “or equals” operator includes a defined? check

```ruby
$VERBOSE = true

class WarnMe
  def var
    @var ||= 42
  end
end

p WarnMe.new.var
```

42
Shortcut Variable Interpolation

The braces are optional with a sigil toting variable

```ruby
@instance = :instance
@@class   = :class
$global   = :global

puts "#@instance, @@class, and #$global variables don't need braces."
```
**Shortcut Variable Interpolation**

The braces are optional with a sigil toting variable

```ruby
@instance = :instance
@@class  = :class
$global  = :global

puts "#@instance, @@class, and #$global variables don't need braces."
```
The braces are optional with a sigil toting variable
Variables From a Regex

It must be a literal regex and it must be on the left side of the match operator

```ruby
if /\A(?<last>\w+),\s*(?<first>\w+)\z/ =~ "Gray, James"
  puts "#{first} #{last}"
end
```

#32 From Michael Foley
Variables From a Regex

It must be a literal regex and it must be on the left side of the match operator

```ruby
if /\A(?<last>\w+),\s*(?<first>\w+)\z/ =~ "Gray, James"
  puts "#{first} #{last}"
end
```
Variables From a Regex

It must be a literal regex and it must be on the left side of the match operator

```
if /\A(?<last>\w+),\s*(?<first>\w+)\z/ =~ "Gray, James"
  puts "#{first} #{last}"
end
```

James Gray
The Unused Variable

Ruby supports the convention of _ as an unused variable name
[ ["James", "Gray", 36],
["Dana", "Gray", 37],
["Summer", "Gray", 2] ].each do |name, ignore, ignore|
  puts name
end

The Unused Variable

Ruby supports the convention of _ as an unused variable name
Ruby supports the convention of _ as an unused variable name.
The Unused Variable

Ruby supports the convention of _ as an unused variable name
Ruby supports the convention of _ as an unused variable name.
Ruby supports the convention of _ as an unused variable name.
Data Structures
Objects Can Contain Themselves

Array Inception

```ruby
ring = [:one, [:two, [:three]]]
ring.last.last <<= ring

position = ring
4.times do
  puts position.first
  position = position.last
end
```
Objects Can Contain Themselves

Array Inception

#34 From Nathaniel Barnes
Objects Can Contain Themselves

Array Inception

#34 From Nathaniel Barnes
Or Just Use `cycle()`

Ruby 1.9 has a nice new repeating iterator

```
ring = %w[one two three].cycle
puts ring.take(4)
```
Or Just Use `cycle()`

Ruby 1.9 has a nice new repeating iterator

```ruby
ring = %w[one two three].cycle
puts ring.take(4)
```
Or Just Use `cycle()`

Ruby 1.9 has a nice new repeating iterator
Or Just Use `cycle()`

Ruby 1.9 has a nice new repeating iterator
Associative Arrays

An ordered (by your criteria) Hash-like thing

```ruby
aa = [ %w[James Gray],
      %w[Yukihiro Matsumoto] ]

p aa.assoc("James")
p aa.rassoc("Matsumoto")
```
Associative Arrays

An ordered (by your criteria) Hash-like thing

```ruby
aa = [ %w[James Gray],
      %w[Yukihiro Matsumoto] ]

p aa.assoc("James")
p aa.rassoc("Matsumoto")
```
Associative Arrays

An ordered (by your criteria) Hash-like thing

```ruby
aa = %w[James Gray, Yukihiro Matsumoto]
p aa.assoc("James")
p aa.rassoc("Matsumoto")
```

```ruby
["James", "Gray"]
["Yukihiro", "Matsumoto"]
```
More Than Two Fields

Sneak some extra data through on the end

```ruby
many_fields = [ %w[James Gray Developer JEG2],
                %w[Yukihiro Matsumoto Language\ Designer yukihiro_matz] ]

first, last, title, twitter = many_fields.assoc("Yukihiro")
puts "matz is a #{title}"

first, last, title, twitter = many_fields.rassoc("Gray")
puts "I am a #{title}"
Many fields = [ %w[James Gray Developer JEG2],
               %w[Yukihiro Matsumoto Language\ Designer yukihiro_matz] ]

First, last, title, twitter = many_fields.assoc(“Yukihiro”)
puts “matz is a #{title}”

First, last, title, twitter = many_fields.rassoc(“Gray”)
puts “I am a #{title}”

More Than Two Fields
Sneak some extra data through on the end
More Than Two Fields

Sneak some extra data through on the end

```ruby
many_fields = [ %w[James Gray Developer JEG2],
                %w[Yukihiro Matsumoto Language\ Designer yukihiro_matz] ]

first, last, title, twitter = many_fields.assoc("Yukihiro")
puts "matz is a #{title}"

first, last, title, twitter = many_fields.rassoc("Gray")
puts "I am a #{title}"```
many_fields = [ %w[James Gray Developer JEG2],
               %w[Yukihiro Matsumoto Language\ Designer yukihiro_matz] ]

first, last, title, twitter = many_fields.assoc("Yukihiro")
puts "matz is a #{title}"

first, last, title, twitter = many_fields.rassoc("Gray")
puts "I am a #{title}"
With Versioned Data

This data structure naturally handles versioning

dana = [[:first, "Dana"], [:last, "Payne"]]

maiden = dana.assoc(:last).last
puts "Dana's maiden name was #{maiden}.

dana.unshift([:last, "Gray"])
married = dana.assoc(:last).last
puts "Dana's married name is #{married}.

current = dana.assoc(:last)
previous = dana[(dana.index(current) + 1)..-1].assoc(:last).last
puts "Dana's previous last name was #{previous}."
With Versioned Data

This data structure naturally handles versioning

#38 From David Brady
With Versioned Data

This data structure naturally handles versioning
dana = [[:first, "Dana"], [:last, "Payne"]]

maiden = dana.assoc(:last).last
puts "Dana's maiden name was #{maiden}.

dana.unshift([:last, "Gray"])
married = dana.assoc(:last).last
puts "Dana's married name is #{married}.

current = dana.assoc(:last)
previous = dana[(dana.index(current) + 1)..''].assoc(:last).last
puts "Dana's previous last name was #{previous}.

Dana's maiden name was Payne.
Dana's married name is Gray.
Dana's previous last name was Payne.

With Versioned Data

This data structure naturally handles versioning
Easy, Fast Memoization

Ruby’s Hash is a memoization algorithm in disguise

```ruby
fibonacci = Hash.new { |numbers, index|
  numbers[index] = fibonacci[index - 2] + fibonacci[index - 1]
}.update(0 => 0, 1 => 1)

p fibonacci[300]
```
Easy, Fast Memoization

Ruby’s Hash is a memoization algorithm in disguise

```ruby
fibonacci = Hash.new { |numbers, index|
  numbers[index] = fibonacci[index - 2] + fibonacci[index - 1]
}.update(0 => 0, 1 => 1)
p fibonacci[300]
```
Easy, Fast Memoization

Ruby’s Hash is a memoization algorithm in disguise

#39 From Benjamin Fleischer
deep = Hash.new { |hash, key| hash[key] = Hash.new(&hash.default_proc) }

deep[:a][:b][:c] = 42
p deep
Autovivification

It’s not just for Perl programs

#40 From Adam Williams
Autovivification

It’s not just for Perl programs

#40 From Adam Williams
```ruby
deep = Hash.new { |hash, key| hash[key] = Hash.new(&hash.default_proc) }

deep[:a][:b][:c] = 42
p deep
```

```
{:a=>{:b=>{:c=>42}}}
```

**Autovivification**

It’s not just for Perl programs

---

#40 From Adam Williams
Rolling Your Own Dispatch Table

Works great with the new Hash and lambda syntax

operations = { number: ->(n) { n.to_i },
              unary_op: ->(op, n) { n.send("#{op}@") },
              binary_op: ->(op, l, r) { l.send(op, r) } } 

stack = []

loop do
  puts stack.map.with_index { |n, i| "#{i}: #{n}" }
  print ">> "
  line = $stdin.gets or break
  type = case line
         when %r{\A[-+*/]\Z} then :binary_op
         when %r{\An\Z} then :unary_op
         else :number
         end
  op = operations[type]
  stack << op[line.strip.tr( 'n', ' '), *stack.pop(op.arity - 1)]
end
Rolling Your Own Dispatch Table

Works great with the new Hash and lambda syntax

#41 From Dave Copeland

```
op = operations[type]
stack << op[line.strip.tr('n', '-'), *stack.pop(op.arity - 1)]
```

```
operations = {  
  number: ->(n) { n.to_i },
  unary_op: ->(op, n) { n.send("#{op}@") },
  binary_op: ->(op, l, r) { l.send(op, r) }
}
stack = []

loop do
  puts stack.map.with_index { |n, i| "#{i}: #{n}" }
  print ">> "
  line = $stdin.gets or break
  type = case line
    when %r{\A[-*/]\Z} then :binary_op
    when %r{\An\Z} then :unary_op
    else :number
  end
  op = operations[type]
  stack << op[line.strip.tr('n', '-'), *stack.pop(op.arity - 1)]
end
```
Rolling Your Own Dispatch Table

Works great with the new Hash and lambda syntax

#41 From Dave Copeland
Rolling Your Own Dispatch Table

Works great with the new Hash and lambda syntax

#41 From Dave Copeland
class Input; def initialize(input) @input = input end end
class Number < Input; def calculate() @input.to_i end end
class UnaryOperation < Input; def calculate(n) n.send("#@input@") end end
class BinaryOperation < Input; def calculate(l, r) l.send(@input, r) end end

stack = [ ]
loop do
  puts stack.map.with_index { |n, i| "#{i}: #{n}" }
  print ">> "
  line = $stdin.gets or break
  type = case line
         when %r{\A[-+*/]\Z} then BinaryOperation
         when %r{\An\Z} then UnaryOperation
         else Number
     end
  op = type.new(line.strip.tr('n', '-'))
  stack << op.calculate(*stack.pop(op.method(:calculate).arity))
end

Or Just Use Ruby to Dispatch

This can be faster than a case statement and calling a lambda

#42 From Josh Susser
Or Just Use Ruby to Dispatch

This can be faster than a case statement and calling a lambda

#42 From Josh Susser
Or Just Use Ruby to Dispatch

This can be faster than a case statement and calling a lambda
Fetching Data

This one method packs so many tricks is hard not to love it

```javascript
params = {var: 42}

p params.fetch(:var)
p params.fetch(:missing, 42)
p params.fetch(:missing) { 40 + 2 }
params.fetch(:missing)
```
Fetching Data

This one method packs so many tricks is hard not to love it

```javascript
params = {var: 42}
params.fetch(:var)
params.fetch(:missing, 42)
params.fetch(:missing) { 40 + 2 }
params.fetch(:missing)
```
Fetching Data

This one method packs so many tricks is hard not to love it

#43 From Michael Foley
Fetching Data

This one method packs so many tricks is hard not to love it
Fetching Data

This one method packs so many tricks is hard not to love it
str = "Price $24.95"

p str[/$\d+(?:\.\d+)?/]  

p str[/$\$(\d+)(?:\.\d+)?/, 1]  

p str[/$(?<dollars>\d+)(?:\.\d+)?/,, :dollars]  

Indexing Into a String by Regex

I use this more often than the match operator

#44 From Jacob Tjørnholm
I use this more often than the match operator

```python
str = "Price $24.95"

p str[/$\d+(?:\.\d+)?/]
op str[/$\d+(?:\.\d+)?/, \d+]
op str[/$\d+(\d+)(?:\.\d+)?/, \d+]
op str[/$\d+<dollars>\d+(?:\.\d+)?/, dollars]
```
Indexing Into a String by Regex

I use this more often than the match operator

```plaintext
str = "Price $24.95"

p str[/$\d+(?::\.\d+)?/]  
p str[/$\d+\d+/\,1]  
p str[/$(?<dollars>\d+)\d+/, :dollars]
```
I use this more often than the match operator

```python
str = "Price $24.95"

p str[\/$\d+(?:\./\d+)?/]

p str[\/$\d+(?:\./\d+)?/], 1
p str[\/$(<dollars>\d+)(?:\./\d+)?/], :dollars

"$24.95"
"24"
"24"
```

Indexing Into a String by Regex
str = "$24.95 per seat"
str[/\$\d+(?:\.\d+)/] = "$9.99"
puts str
str[/\$\d+(?:\.\d+)(\b)/, 1] = " USD"
puts str

You Can Even Assign by Regex
I don’t tend to use this version

#46 From Jacob Tjørnholm
You Can Even Assign by Regex

I don’t tend to use this version

#46 From Jacob Tjørnholm
You Can Even Assign by Regex

I don’t tend to use this version

---

#46 From Jacob Tjørnholm
str = "$24.95 per seat"

str[//\$\d+(?:\.\d+)/] = "$9.99"
puts str

str[//\$\d+(?:\.\d+)(\b)/, 1] = " USD"
puts str

$9.99 per seat
$9.99 USD per seat

You Can Even Assign by Regex

I don’t tend to use this version

#46 From Jacob Tjørnholm
Finding the Last Match

Work from the right instead of the left

```python
expression = "40 + 2 = 42"

p expression[expression.rindex(\d+)..-1]
p expression[expression.rindex(\b\d+)..-1]
```
Finding the Last Match

Work from the right instead of the left
Finding the Last Match

Work from the right instead of the left

define expression = "40 + 2 = 42"

print(expression[expression.rindex(\d+)\-1])
print(expression[expression.rindex(\b\d+)\-1])

"2"
"42"
Chew Through Binary Data

These methods are also useful on old “fixed width” data files

#47 From Benjamin Fleischer
width, height = ARGF.read(24).unpack("@16N2")
puts "Width: #{width} pixels"
puts "Height: #{height} pixels"

Chew Through Binary Data
These methods are also useful on old “fixed width” data files

#47 From Benjamin Fleischer
```
width, height = ARGF.read(24).unpack("@16N2")
puts " Width:  #{width} pixels"
puts " Height:  #{height} pixels"
```

```
$ ruby png_size.rb rails_project/public/images/rails.png
  Width:  50 pixels
  Height:  64 pixels
```
Iterators
Iterating in Lockstep

This can walk through two or more collections at once

```ruby
letters = "a"..'d'
numbers = 1..3

letters.zip(numbers) do |letter, number|
  p(letter: letter, number: number)
end
```
Iterating in Lockstep

This can walk through two or more collections at once

```ruby
letters = "a"..'d'
numbers = 1..3

letters.zip(numbers).each do |letter, number|
  p(letter: letter, number: number)
end
```
Iterating in Lockstep

This can walk through two or more collections at once

letters = "a"..'d'
numbers = 1..3

letters.zip(numbers).each do |letter, number|
p(letter: letter, number: number)
end

{ :letter => "a", :number => 1 }
{ :letter => "b", :number => 2 }
{ :letter => "c", :number => 3 }
{ :letter => "d", :number => nil }

#48 From Benjamin Fleischer
Partition Your Data

This iterator has been in Ruby a long time now, but I seldom see it used

```ruby
Person = Struct.new(:name, :gender)
people = [ Person.new("James", :male),
          Person.new("Dana", :female),
          Person.new("Summer", :female) ]

males, females = people.partition { |person| person.gender == :male }

puts "Males:", males.map { |male| " #{male.name}" }
puts "Females:", females.map { |female| " #{female.name}" }
```

---

#49 From Piotr Szotkowski
Partition Your Data

This iterator has been in Ruby a long time now, but I seldom see it used.

#49 From Piotr Szotkowski
Person = Struct.new(:name, :gender)
people = [ Person.new("James", :male),
          Person.new("Dana", :female),
          Person.new("Summer", :female) ]

males, females = people.partition { |person| person.gender == :male }

puts "Males: ", males.map { |male| " #{male.name}" }
puts "Females: ", females.map { |female| " #{female.name}" }

Males: 
  James
Females: 
  Dana
  Summer

Partition Your Data

This iterator has been in Ruby a long time now, but I seldom see it used

#49 From Piotr Szotkowski
Take Data in Chunks

A newer iterator for us to exploit

#50 From Piotr Szotkowski
Take Data in Chunks

A newer iterator for us to exploit

#50 From Piotr Szotkowski
Take Data in Chunks

A newer iterator for us to exploit

#50 From Piotr Szotkowski
map() + flatten() = flat_map()

This can be a handy shortcut for working with nested collections
map() + flatten() = flat_map()

This can be a handy shortcut for working with nested collections
require "pp"

chess_squares = ("A"..'H').flat_map { |column|
  (1..8).map { |row| "#{column}#{row}" }
}

pp chess_squares

\texttt{map()} + \texttt{flatten()} = \texttt{flat\_map()}

This can be a handy shortcut for working with nested collections
Replace Ugly inject() Calls

This iterator was added to kill a bad yet common usage of inject()

```ruby
# instead of:  (1..3).inject({ }) { |hash, n| hash[n] = true; hash }
object = (1..3).each_with_object({ }) do |n, hash|
  hash[n] = true
end
p object
```
Replace Ugly inject() Calls

This iterator was added to kill a bad yet common usage of inject()
Replace Ugly inject() Calls

This iterator was added to kill a bad yet common usage of inject()

# instead of: (1..3).inject({ }) { |hash, n| hash[n] = true; hash }
object = (1..3).each_with_object({ }) do |n, hash|
  hash[n] = true
end
p object
Replace Ugly inject() Calls

This iterator was added to kill a bad yet common usage of inject()

```ruby
# instead of:  (1..3).inject({ }) { |hash, n| hash[n] = true; hash }
object = (1..3).each_with_object({ }) do |n, hash|
  hash[n] = true
end
p object

{1=>true, 2=>true, 3=>true}
```
numbers = 1..10

p numbers.take(3)
p numbers.drop(7)
p numbers.take_while { |n| n <= 5 }  
p numbers.drop_while { |n| n <= 5 }

Take a Little Off the Top
Pull or skip from the beginning of a list
numbers = 1..10

p numbers.take(3)
p numbers.drop(7)
p numbers.take_while { |n| n <= 5 }  
p numbers.drop_while { |n| n <= 5 }

Take a Little Off the Top

Pull or skip from the beginning of a list
Take a Little Off the Top

Pull or skip from the beginning of a list

```ruby
numbers = 1..10

p numbers.take(3)
p numbers.drop(7)
p numbers.take_while { |n| n <= 5 }
p numbers.drop_while { |n| n <= 5 }
```
Take a Little Off the Top

Pull or skip from the beginning of a list

```ruby
numbers = 1..10
p numbers.take(3)
p numbers.drop(7)
p numbers.take_while { |n| n <= 5 }
p numbers.drop_while { |n| n <= 5 }
```
Take a Little Off the Top

Pull or skip from the beginning of a list

numbers = 1..10
p numbers.take(3)
p numbers.drop(7)
p numbers.take_while { |n| n <= 5 }
p numbers.drop_while { |n| n <= 5 }

[1, 2, 3]
[8, 9, 10]
[1, 2, 3, 4, 5]
[6, 7, 8, 9, 10]
Manual Iteration

You can step through the new Enumerator objects by hand

row = %w[odd even].cycle

puts "<table>
("A"..'E').each do |letter|
  puts %Q{  <tr class="#{row.next}">
    <td>#{letter}</td>
  </tr>}
end
puts "</table>"
Manual Iteration

You can step through the new Enumerator objects by hand

```ruby
row = %w[odd even].cycle

puts "<table>"
("A".."E").each do |letter|
  puts %Q{ <tr class="#{row.next}"><td>#{letter}</td></tr> }
end
puts "</table>"
```
Manual Iteration

You can step through the new Enumerator objects by hand

```ruby
row = %w[odd even].cycle

puts "<table>"
("A"..'E').each do |letter|
  puts %Q{ <tr class="#{row.next}"><td>#{letter}</td></tr> }
end
puts "</table>"
```
You can step through the new Enumerator objects by hand:

```ruby
row = %w[odd even].cycle
puts "<table>"
("A"..'E').each do |letter|
  puts %Q{ <tr class="#{row.next}"><td>#{letter}</td></tr> }
end
puts "</table>"
```

```html
table
  <tr class="odd"><td>A</td></tr>
  <tr class="even"><td>B</td></tr>
  <tr class="odd"><td>C</td></tr>
  <tr class="even"><td>D</td></tr>
  <tr class="odd"><td>E</td></tr>
</table>
```
A Smarter loop

Or get some help from loop()
animals = %w[cat bat rat]

enum = animals.to_enum
3.times do
  enum.next
end

enum.next rescue puts "Error raised: #{$!.class}"

enum.rewind
loop do
  puts "Processing #{enum.next}..."
end

A Smarter loop()

Or get some help from loop()
animals = %w[cat bat rat]

enum = animals.to_enum
3.times do
  enum.next
end
enum.next rescue puts "Error raised: $#{Exception.class}"

enum.rewind
loop do
  puts "Processing #{enum.next}..."
end

A Smarter loop

Or get some help from loop()
A Smarter loop()

Or get some help from loop()
Chaining Iterators

Enumerators also allow for the chaining of iteration

#56 From David Brady
Chaining Iterators

Enumerators also allow for the chaining of iteration

```ruby
p ("a"..'z').each_cons(3)
  .map(&:join)
  .select { |letters| letters =~ /[aeiouy]/ }
```
Chaining Iterators

Enumerators also allow for the chaining of iteration
Add an Index to Any Iterator

Enumerator supports this special method just for chaining

#57

```ruby
votes = { "Josh" => %w[SBPP POODR GOOS],
          "Avdi"  => %w[POODR SBPP GOOS],
          "James" => %w[POODR GOOS SBPP],
          "David" => %w[GOOS SBPP POODR],
          "Chuck" => %w[GOOS POODR SBPP] }

tally = Hash.new(0)
votes.values.each do |personal_selections|
  personal_selections.each_with_object(tally).with_index do |(vote, totals), i|
    totals[vote] += personal_selections.size - i
  end
end

p tally
```
Add an Index to Any Iterator

Enumerator supports this special method just for chaining
Add an Index to Any Iterator

Enumerator supports this special method just for chaining

```ruby
votes = { "Josh" => %w[SBPP POODR GOOS],
          "Avdi" => %w[POODR SBPP GOOS],
          "James" => %w[POODR GOOS SBPP],
          "David" => %w[GOOS SBPP POODR],
          "Chuck" => %w[GOOS POODR SBPP] }

tally = Hash.new(0)
votes.values.each do |personal_selections|
  personal_selections.each_with_object(tally).with_index do |(vote, totals), i|
    totals[vote] += personal_selections.size - i
  end
end

p tally
```
Add an Index to Any Iterator

Enumerator supports this special method just for chaining

```ruby
votes = { "Josh" => %w{SBPP POODR GOOS},
         "Avdi"  => %w{POODR SBPP GOOS},
         "James" => %w{POODR GOOS SBPP},
         "David" => %w{GOOS SBPP POODR},
         "Chuck" => %w{GOOS POODR SBPP} }

tally = Hash.new(0)
votes.values.each do |personal_selections|
  personal_selections.each_with_object(tally).with_index do |(vote, totals), i|
    totals[vote] += personal_selections.size - i
  end
end
p tally

{"SBPP"=>9, "POODR"=>11, "GOOS"=>10}
```
Intermission
Why Do This?

- We trade in the currency of ideas
- A bad plan is better than no plan
- It’s good to have an appreciation for how rich Ruby is, as a language
I Don’t Have Time for This!
Core Ruby
def build_class(parent, extra_methods = { })
    Class.new(parent) do
        extra_methods.each do |name, result|
            body = result.is_a?(Proc) ? result : -> { result }
            define_method(name, &body)
        end
    end
end

Thingy = build_class(Object, value: 42, dynamic: -> { :called })
thingy = Thingy.new
p thingy.value
p thingy.dynamic

Programmatically Build Classes
This allows your code to construct what is needed

#58 From Benjamin Fleischer
def build_class(parent, extra_methods = { })
    Class.new(parent) do
        extra_methods.each do |name, result|
            body = result.is_a?(Proc) ? result : -> { result }
            define_method(name, &body)
        end
    end
end

Thingy = build_class(Object, value: 42, dynamic: -> { :called })
thingy = Thingy.new
p thingy.value
p thingy.dynamic

Programmatically Build Classes
This allows your code to construct what is needed
def build_class(parent, extra_methods = { })
    Class.new(parent) do
        extra_methods.each do |name, result|
            body = result.is_a?(Proc) ? result : -> { result }
            define_method(name, &body)
        end
    end
end

Thingy = build_class(Object, value: 42, dynamic: -> { :called })
thingy = Thingy.new
p thingy.value
p thingy.dynamic

42
:called

Programmatically Build Classes
This allows your code to construct what is needed

#58 From Benjamin Fleischer
def build_module(extra_methods = { })
    Module.new do
        extra_methods.each do |name, result|
            body = result.is_a?(Proc) ? result : -> { result }
            define_method(name, &body)
        end
    end
end

Mixin = build_module(value: 42, dynamic: -> { :called })
thingy = Object.new.extend(Mixin)
p thingy.value
p thingy.dynamic

Programmatically Build Modules

Modules can also be constructed by code

#59 From Benjamin Fleischer
def build_module(extra_methods = { })
    Module.new do
        extra_methods.each do |name, result|
            body = result.is_a?(Proc) ? result : -> { result }
            define_method(name, &body)
        end
    end
end

Mixin = build_module(value: 42, dynamic: -> { :called })
thingy = Object.new.extend(Mixin)
p thingy.value
p thingy.dynamic

Programmatically Build Modules

Modules can also be constructed by code

#59 From Benjamin Fleischer
Programmatically Build Modules

Modules can also be constructed by code
Inherit From an Expression
Programmatically build parent classes

#60 From Benjamin Fleischer
def Value(*fields)
    Class.new do
        define_method(:initialize) do |*args|
            fail ArgumentError, "wrong argument count" unless args.size == fields.size
            fields.zip(args) do |field, arg|
                instance_variable_set("@#{field}", arg)
            end
            end
        fields.each do |field|
            define_method(field) { instance_variable_get("@#{field}") }
        end
    end
end

class Name < Value(:first, :last)
    def full
        "#{first} #{last}"
    end
end

james = Name.new("James", "Gray")
puts james.full
Inherit From an Expression

Programmatically build parent classes

#60 From Benjamin Fleischer

def Value(*fields)
    Class.new do
        define_method(:initialize) do |*args|
            fail ArgumentError, "wrong argument count" unless args.size == fields.size
            fields.zip(args) do |field, arg|
                instance_variable_set("@#{field}", arg)
            end
        end
        fields.each do |field|
            define_method(field) { instance_variable_get("@#{field}") }
        end
    end
end

class Name < Value(:first, :last)
    def full
        "#{first} #{last}"
    end
end

james = Name.new("James", "Gray")
puts james.full

James Gray
def LimitedUse(limit)
    Module.new do
        define_singleton_method(:included) do |parent|
            count = 0
            parent.public_instance_methods.each do |method|
                define_method(method) do |*args|
                    fail "Over use limit" if (count += 1) > limit
                super(*args)
            end
        end
    end
end

class ShortLived
    include LimitedUse(3)
end

limited = ShortLived.new
puts Array.new(3) { limited.to_s }
limited.to_s

Mix-in an Expression
Module inclusion works the same way
def LimitedUse(limit)
    Module.new do
        define_singleton_method(:included) do |parent|
            count = 0
            parent.public_instance_methods.each do |method|
                define_method(method) do |*args|
                    fail "Over use limit" if (count += 1) > limit
                    super(*args)
                end
            end
        end
    end
end

class ShortLived
    include LimitedUse(3)
end

limited = ShortLived.new
puts Array.new(3) { limited.to_s }
limited.to_s
Mix-in an Expression

Module inclusion works the same way
class LimitedUse < Module
  def initialize(limit)
    @limit = limit
    super do
      define_singleton_method(:included) do |parent|
        count = 0
        parent.public_instance_methods.each do |method|
          define_method(method) do |*args|
            fail "Over use limit" if (count += 1) > limit
            super(*args)
          end
        end
      end
      end
      end
    end
def to_s; "LimitedUse.new(#{@limit})" end
end
class ShortLived; include LimitedUse.new(3) end
p ShortLived.ancestors
limited = ShortLived.new
puts Array.new(3) { limited.to_s }
limited.to_s

Subclass Module
It’s possible to create modules with state

#62 From Avdi Grimm
It’s possible to create modules with state

```ruby
class LimitedUse < Module
  def initialize(limit)
    @limit = limit
    super do
      define_singleton_method(:included) do |parent|
        count = 0
        parent.public_instance_methods.each do |method|
          define_method(method) do |*args|
            fail "Over use limit" if (count += 1) > limit
            super(*args)
          end
        end
      end
    end
    def to_s; "LimitedUse.new(#{@limit})" end
  end
end
class ShortLived; include LimitedUse.new(3) end
p ShortLived.ancestors
limited = ShortLived.new
puts Array.new(3) { limited.to_s }
limited.to_s
```
Subclass Module

It’s possible to create modules with state

#62 From Avdi Grimm
Subclass Module

It’s possible to create modules with state

#62 From Avdi Grimm
Empty Types

You don’t need a full declaration to define a class or module

#63 From Steve Klabnik
Empty Types

You don’t need a full declaration to define a class or module
module MyNamespace
    module Errors
        # instead of: class MyNamespaceError < RuntimeError; end
        MyNamespaceError = Class.new(RuntimeError)
        WhateverError = Class.new(MyNamespaceError)
    end
end

p MyNamespace::Errors::WhateverError.ancestors

[MyNamespace::Errors::WhateverError,
 MyNamespace::Errors::MyNamespaceError,
 RuntimeError, StandardError, Exception, Object, Kernel, BasicObject]
Don’t Inherit From Struct

Struct takes a block

#64 From Chris Hunt
Don’t Inherit From Struct

Struct takes a block

#64 From Chris Hunt
Don’t Inherit From Struct

Struct takes a block

#64 From Chris Hunt
# instead of: class Name < Struct.new(:first, :last) ... end
Struct.new("Name", :first, :last) do
  def full
    "#{first} #{last}"
  end
end

james = Struct::Name.new("James", "Gray")
puts james.full

Struct Without the Assignment

Subclasses of Struct can be placed under that namespace
Subclasses of Struct can be placed under that namespace

```ruby
# instead of:  class Name < Struct.new(:first, :last) ... end
Struct.new("Name", :first, :last) do
  def full
    "#{first} #{last}"
  end
end

james = Struct::Name.new("James", "Gray")
puts james.full
```
Subclasses of Struct can be placed under that namespace

```
# instead of:  class Name < Struct.new(:first, :last) ... end
Struct.new("Name", :first, :last) do
  def full
    "#{first} #{last}"
  end
end

james = Struct::Name.new("James", "Gray")
puts james.full
```
Struct Without the Assignment

Subclasses of Struct can be placed under that namespace
class Greeter
  GREETING_REGEX = /\Aaloha_\w+/z/

  def method_missing(method, *args, &block)
    if method =~ GREETING_REGEX
      "Aloha #{method.to_s.split("_")[1..-1].map(&:capitalize).join(" ")}
    else
      super
    end
  end

  def respond_to_missing?(method, include_private = false)
    method =~ GREETING_REGEX
  end
end

greeter = Greeter.new
p greeter.respond_to?(:aloha_james_gray)
puts greeter.method(:aloha_james_gray).call

Be a Good method_missing() User
Fix respond_to?() and method() for your method_missing() usage

#66 From Mike Burns
class Greeter
  GREETING_REGEX = /\Aaloha_\w+/\z/

  def method_missing(method, *args, &block)
    if method =~ GREETING_REGEX
      "Aloha #{method.to_s.split("_")[1-1].map(&:capitalize).join(" ")}
    else
      super
    end
  end

  def respond_to_missing?(method, include_private = false)
    method =~ GREETING_REGEX
  end
end

greeter = Greeter.new
p greeter.respond_to?(:aloha_james_gray)
puts greeter.method(:aloha_james_gray).call

Be a Good method_missing() User

Fix respond_to?() and method() for your method_missing() usage

#66 From Mike Burns
class Greeter
    GREETING_REGEX = /\Aloha\_\w+\z/

    def method_missing(method, *args, &block)
        if method =~ GREETING_REGEX
            "Aloha #{method.to_s.split("_")[1..-1].map(&:capitalize).join(" ")}
        else
            super
        end
    end

    def respond_to_missing?(method, include_private = false)
        method =~ GREETING_REGEX
    end
end

greeter = Greeter.new
p greeter.respond_to?(:aloha_james_gray)
puts greeter.method(:aloha_james_gray).call

---

Be a Good method_missing() User

Fix respond_to?() and method() for your method_missing() usage

#66 From Mike Burns
Be a Good method_missing() User

Fix respond_to?() and method() for your method_missing() usage

#66 From Mike Burns
class Whatever
  def self.peek_inside(&block)
    define_method(:peek, &block)
    method = instance_method(:peek)
    remove_method(:peek)
    method
  end

  def initialize(secret)
    @secret = secret
  end
end

magic_key = Whatever.peek_inside { @secret }
meaning = Whatever.new(42)
other = Whatever.new(:other)

p magic_key.bind(meaning).call
p magic_key.bind(other).call
instance_eval() In Disguise

This trick is faster than the real thing

```ruby
class Whatever
  def self.peek_inside(&block)
    define_method(:peek, &block)
    method = instance_method(:peek)
    remove_method(:peek)
    method
  end

  def initialize(secret)
    @secret = secret
  end
end

magic_key = Whatever.peek_inside { @secret }
meaning = Whatever.new(42)
other = Whatever.new(:other)

p magic_key.bind(meaning).call
p magic_key.bind(other).call
```
This trick is faster than the real thing
instance_eval() In Disguise

This trick is faster than the real thing
Iterating Over Each Object

Remember, we’re talking about MRI here

#68 From Benjamin Fleischer
Iterating Over Each Object

Remember, we’re talking about MRI here
Iterating Over Each Object

Remember, we’re talking about MRI here

```ruby
ObjectSpace.each_object do |object|
  puts object if object.is_a? String
end
```

```ruby
.ext
  CONFIG["PREP"] = "miniruby$(EXEEXT)"
EXOUT
no
LIBRUBY_RELATIVE
ARCHFILE
EXECUTABLE_EXTS
nodoc
...
```
Iterating Over Specific Types

Focus in on specific objects

#69 From Benjamin Fleischer
Iterating Over Specific Types

Focus in on specific objects

#69 From Benjamin Fleischer
Iterating Over Specific Types

Focus in on specific objects

#69 From Benjamin Fleischer
Count All Objects

You don’t need to iterate if you just want counts

```ruby
require "pp"
pp ObjectSpace.count_objects
```
Count All Objects

You don’t need to iterate if you just want counts

```ruby
require "pp"
pp ObjectSpace.count_objects
```
Count All Objects

You don’t need to iterate if you just want counts

#70 Benjamin Fleischer
Profile the Garbage Collector

This can help find memory leaks

#71 From Peter Cooper
Profile the Garbage Collector

This can help find memory leaks

#71 From Peter Cooper
Profile the Garbage Collector

This can help find memory leaks
Profile the Garbage Collector

This can help find memory leaks

#71 From Peter Cooper
Tapping Into a Call Chain

Also used to mutate without affecting the return value

```ruby
puts "sssssssssssssemaj".sub(/\A(\w)\1+/ , '\1')
  .reverse
  .capitalize
```
Tapping Into a Call Chain

Also used to mutate without affecting the return value

#72 From David Brady
puts "sssssssssssssemaj".sub(/\A(\w)\1+/, '\1')
tap { |str| p str.size }
.reverse
.capitalize
puts "sssssssssssssemaj".sub(/\A([^\w])\1+/, '\1')
.tap { |str| p str.size }
.reverse
.capitalized
puts "sssssssssssemaj".sub(/\A(\w)\1+/, '\1')
.tap { |str| p str.size }
.reverse
.capitalize

Tapping Into a Call Chain
Also used to mutate without affecting the return value

#72 From David Brady
Sneaking in a p() Without tap()

In Ruby 1.9, p() returns its argument
Sneaking in a `p()` Without `tap()`

In Ruby 1.9, `p()` returns its argument

```ruby
puts p("sssssssssssemaj".sub(/\A(\w)\1+/,'\1')).reverse.capitalize
```
Sneaking in a `p()` Without `tap()`

In Ruby 1.9, `p()` returns its argument

```ruby
puts p("sssssssssssssemaj".sub(/\A(\w)\1+/ , '\1')).reverse.capitalize
```

"semaj"
James
Bubbling Up Thread Errors

This can be quite helpful when debugging threaded code

Thread.abort_on_exception = true
Thread.new do
  fail "Oops, we can't continue"
end
loop do
  sleep
end
Bubbling Up Thread Errors

This can be quite helpful when debugging threaded code

```ruby
Thread.abort_on_exception = true

Thread.new do
  fail "Oops, we can't continue"
end

loop do
  sleep
end
```
Bubbling Up Thread Errors

This can be quite helpful when debugging threaded code
The $DEBUG Flag

Turn your debugging code on and off as needed
The `$DEBUG` Flag

Turn your debugging code on and off as needed

```ruby
def var
  @var ||= 40
end

if $DEBUG
  puts "var is %p" % var
end
p var + 2
```
The $DEBUG Flag

Turn your debugging code on and off as needed
The $DEBUG Flag

Turn your debugging code on and off as needed
Two Tests in One

This works on any Comparable object

```ruby
p 2.between?(1, 10)
p "cat".between?("bat", "rat")
```
Two Tests in One

This works on any Comparable object
This works on any Comparable object
Line-by-line Reading

foreach() = open() + each()

#77 From Benjamin Fleischer
Line-by-line Reading

foreach() = open() + each()

#77 From Benjamin Fleischer
Line-by-line Reading

foreach() = open() + each()

#77 From Benjamin Fleischer
The Write-a-file Shortcut

You can even dump some data without a call to open()

```ruby
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")

puts
File.write("output.txt", "one and a half\ntwo\n", 4)
puts File.read("output.txt")

puts
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
```
The Write-a-file Shortcut

You can even dump some data without a call to open()

```ruby
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
puts
File.write("output.txt", "one and a half\ntwo\n", 4)
puts File.read("output.txt")
puts
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
```
The Write-a-file Shortcut

You can even dump some data without a call to open()

```ruby
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
puts
File.write("output.txt", "one and a half\ntwo\n", 4)
puts File.read("output.txt")
puts
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
```
The Write-a-file Shortcut

You can even dump some data without a call to open()

```ruby
File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
puts File.write("output.txt", "one and a half\ntwo")
puts File.read("output.txt")
puts File.write("output.txt", "one\ntwo\nthree\n")
puts File.read("output.txt")
```
Daemonize Your Process

I often see Ruby code roll their own, but it’s built-in now

```ruby
Process.daemon
loop do
  sleep
end
```
Daemonize Your Process

I often see Ruby code roll their own, but it’s built-in now
Daemonize Your Process

I often see Ruby code roll their own, but it’s built-in now

```ruby
Process.daemon
loop do
  sleep
end
```

```
$ ruby daemonize_your_process.rb
$ ps auxww | grep daemonize
james           27071   ... grep daemonize
james           27044   ... ruby daemonize_your_process.rb
james           26564   ... emacs daemonize_your_process.rb
$ kill 27044
$ ps auxww | grep daemonize
james           26564   ... emacs daemonize_your_process.rb
james           27153   ... grep daemonize
```
If you only learn one way to launch a process, let it be spawn()

```ruby
pid = spawn( { "SOME_VAR" => "42"}, # env
             "./child_process.rb", # the command
             in: open(__FILE__) ) # options
Process.wait(pid)
```

```ruby
#!/usr/bin/env ruby -w
puts "SOME_VAR=%p" % ENV["SOME_VAR"]
puts "$stdin.read:", $stdin.read
```

Process Launching on Steroids

If you only learn one way to launch a process, let it be spawn()
Process Launching on Steroids

If you only learn one way to launch a process, let it be spawn()

```ruby
pid = spawn( { "SOME_VAR" => "42" }, # env
"./child_process.rb", # the command
in: open(__FILE__) ) # options
Process.wait(pid)
```

```ruby
#!/usr/bin/env ruby -w
puts "SOME_VAR=%p" % ENV["SOME_VAR"]
puts "$stdin.read:", $stdin.read
```
Process Launching on Steroids

If you only learn one way to launch a process, let it be `spawn()`

```ruby
pid = spawn( { "SOME_VAR" => "42" }, # env
              "/child_process.rb", # the command
             in: open(__FILE__) ) # options
Process.wait(pid)
```

```ruby
#!/usr/bin/env ruby -w
.puts "SOME_VAR=%p" % ENV["SOME_VAR"]
.puts "$stdin.read: ", $stdin.read
```

# 80
Process Launching on Steroids

If you only learn one way to launch a process, let it be `spawn()`

```ruby
pid = spawn( { "SOME_VAR" => "42" }, # env
"./child_process.rb", # the command
  in: open(__FILE__) ) # options
Process.wait(pid)
```

```ruby
#!/usr/bin/env ruby -w

puts "SOME_VAR=%p" % ENV["SOME_VAR"]
puts "$stdin.read:", $stdin.read
```

if you only learn one way to launch a process, let it be spawn()
If you only learn one way to launch a process, let it be spawn()

```ruby
#!/usr/bin/env ruby -w
puts "SOME_VAR=%p" % ENV[SOME_VAR]
puts "$stdin.read:", $stdin.read
```

```bash
$ ruby process_launching_on_steroids.rb
SOME_VAR="42"
$stdin.read:
```

```ruby
pid = spawn( {"SOME_VAR" => "42"},  # env
              "./child_process.rb",  # the command
              in: open(__FILE__) )   # options
Process.wait(pid)
```

---

Process Launching on Steroids

If you only learn one way to launch a process, let it be spawn()
Seriously, spawn()

This sucker has all the features you need
spawn({"A_VAR" => "Whatever"}, cmd)    # set an environment variable
spawn({"A_VAR" => nil}, cmd)        # clear an environment variable
spawn(env, cmd, unsetenv_others: true)    # clear unset environment variables

spawn("rm *.txt")                    # normal shell expansion
spawn("rm", "*.txt")                 # bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.txt")    # rename command in process list

spawn(cmd, pgroup: 0)                 # change the process group
spawn(cmd, rlimit_cpu: Process.getrlimit(:CPU).last) # raise resource limits
spawn(cmd, chdir: path)               # change working directory
spawn(cmd, umask: 0222)               # change permissions

spawn(cmd, in: io)                    # redirect IO
spawn(cmd, io => [open, args])
spawn(cmd, io => :close)              # close an IO
spawn(cmd, close_others: true)       # close unset IO

pid = spawn(cmd); Process.detach(pid) # asynchronous
pid = spawn(cmd); Process.wait(pid)   # synchronous

Seriously, spawn()
This sucker has all the features you need
Seriously, spawn()

This sucker has all the features you need
Seriously, spawn()

This sucker has all the features you need
Seriously, spawn()

This sucker has all the features you need
Seriously, spawn()

This sucker has all the features you need
The Standard Library
Get Random Data

Ruby can generate random data for you

```ruby
require "securerandom"

p SecureRandom.random_number
p SecureRandom.random_number(100)
puts
p SecureRandom.hex(20)
p SecureRandom.base64(20)
p SecureRandom.urlsafe_base64(20)
p SecureRandom.random_bytes(20)
puts
p SecureRandom.uuid
```
require "securerandom"

p SecureRandom.random_number
p SecureRandom.random_number(100)
puts
p SecureRandom.hex(20)
p SecureRandom.base64(20)
p SecureRandom.urlsafe_base64(20)
p SecureRandom.random_bytes(20)
puts
p SecureRandom.uuid

Get Random Data

Ruby can generate random data for you
Get Random Data

Ruby can generate random data for you

```ruby
require "securerandom"

p SecureRandom.random_number
p SecureRandom.random_number(100)
puts
p SecureRandom.hex(20)
p SecureRandom.base64(20)
p SecureRandom.urlsafe_base64(20)
p SecureRandom.random_bytes(20)
puts
p SecureRandom.uuid
```
Get Random Data

Ruby can generate random data for you

```ruby
require "securerandom"

p SecureRandom.random_number
p SecureRandom.random_number(100)
puts
p SecureRandom.hex(20)
p SecureRandom.base64(20)
p SecureRandom.urlsafe_base64(20)
p SecureRandom.random_bytes(20)
puts
p SecureRandom.uuid
```
Get Random Data

Ruby can generate random data for you

```ruby
require "securerandom"

p SecureRandom.random_number
p SecureRandom.random_number(100)

puts
p SecureRandom.hex(20)
p SecureRandom.base64(20)
p SecureRandom.urlsafe_base64(20)
p SecureRandom.random_bytes(20)

0.8337575534446421
50

"5f1edc66708381c18f5527ccc49c0fad7822d5f3"
"aoRvYu/V6/MeluooKG+tf4yjyyU="
"WfWpdByTVMdliaymKB_FmqTGgE8"
"!d\x94\xE6viS\xCB\xA7\xDB\x84\xCF\x8EY\xCB\I\x9F6\xE6\xFD"

"5243d395-f317-4394-aa5d-762005d1fe7a"
```
Read From the Web

This is the easiest way to pull something off the Web

#83 From Benjamin Fleischer
require "open-uri"

open("http://rubyrogues.com/feed/") do |feed|
  puts feed.read.scan(%r{<title>(\d+\s*RR\b[^<>]*)</title>})
end
require "open-uri"

open("http://rubyrogues.com/feed/") do |feed|
  puts feed.read.scan(%r{<title>(\d+\s*RR\b[^<]*)</title>})
end

Read From the Web

This is the easiest way to pull something off the Web

#83 From Benjamin Fleischer
require "open-uri"

open("http://rubyrogues.com/feed/") do |feed|
  puts feed.read.scan(%r{<title>(\d+\s*RR\b[^<]*)(?<!\>)</title>})
end

073 RR APIs
072 RR Entrepreneurship with Amy Hoy
071 RR Zero Downtime Deploys
070 RR What is a Good Starter Project?
069 RR Therapeutic Refactoring with Katrina Owen
068 RR Book Club: Growing Object Oriented Software Guided by Tests
067 RR Gary Bernhardt’s Testing Style
066 RR Rails Bridge with Sarah Mei
065 RR Functional vs Object Oriented Programming with Michael Feathers
064 RR Presenting at Conferences
...

Read From the Web

This is the easiest way to pull something off the Web

#83 From Benjamin Fleischer
require "shellwords"

p Shellwords.shellwords('one "two" "a longer three"')
p Shellwords.shellwords("one 'two' 'a longer three'")
p Shellwords.shellwords('"escaped "quote" characters"')
p Shellwords.shellwords('escaped\ spaces')
p Shellwords.shellwords(%Q{'back to'" back quoting"})

p Shellwords.shellescape("two words")
p Shellwords.shellescape('"quotes" included')

p Shellwords.shelljoin(["two words", "'quotes" included'"])
require "shellwords"

p Shellwords.shellwords('one "two" "a longer three"')
p Shellwords.shellwords("one 'two' 'a longer three'")
p Shellwords.shellwords("escaped \"quote\" characters")
p Shellwords.shellwords('escaped\ spaces')
p Shellwords.shellwords(%Q{"back to'" back quoting"})

p Shellwords.shellescape("two words")
p Shellwords.shellescape("quotes" included')

p Shellwords.shelljoin(["two words", "quotes" included'])
require "shellwords"

p Shellwords.shellwords('one "two" "a longer three"')
p Shellwords.shellwords("one 'two' 'a longer three'")
p Shellwords.shellwords('"escaped \"quote\" characters"')
p Shellwords.shellwords('escaped\ spaces')
p Shellwords.shellwords(%Q{'back to'" back quoting'})

p Shellwords.shellescape("two words")
p Shellwords.shellescape("quotes" included')
p Shellwords.shelljoin(['"two words", "quotes" included'])

Escape For the Shell

This is a great helper when using Ruby as a glue language
require "shellwords"

p Shellwords.shellwords('one "two" "a longer three"')
p Shellwords.shellwords("one 'two' 'a longer three'")
p Shellwords.shellwords('''escaped "quote" characters''')
p Shellwords.shellwords('escaped\ spaces')
p Shellwords.shellwords(%Q{'back to" back quoting'})

p Shellwords.shellescape("two words")
p Shellwords.shellescape('"quotes" included')

p Shellwords.shelljoin(['"two words", "quotes" included'])
require "shellwords"

Shellwords.shellwords('one "two" "a longer three"
Shellwords.shellwords('one 'two' 'a longer three"
Shellwords.shellwords('escaped "quote" characters"
Shellwords.shellwords('escaped spaces"
Shellwords.shellwords(%Q{"back to back quoting"
"two\ words"
"\\"quotes\\\"\\ included"
"two\\ words \"quotes\\\"\\ included"

Shellwords.shellescape("two words"
Shellwords.shellescape("quotes included"

Shellwords.shelljoin(["two words", "quotes included"])

---

Escape For the Shell

This is a great helper when using Ruby as a glue language
Template Methods

Here’s a feature of ERb that I never see used

#85 From Masatoshi Seki

```ruby
require "erb"

class Name
  def initialize(first, last)
    @first = first
    @last = last
  end
  attr_reader :first, :last
  extend ERB::DefMethod
  def_erb_method("full", "full_name.erb")
  def_erb_method("last_first", "last_name_first.erb")
end

james = Name.new("James", "Gray")
puts james.full
puts james.last_first
```
Template Methods

Here’s a feature of ERb that I never see used

#85 From Masatoshi Seki
#85 From Masatoshi Seki

## Template Methods

Here’s a feature of ERb that I never see used

```ruby
require "erb"

class Name
  def initialize(first, last)
    @first = first
    @last = last
  end
  attr_reader :first, :last
  extend ERB::DefMethod
  def_erb_method("full", "full_name.erb")
  def_erb_method("last_first", "last_name_first.erb")
end

james = Name.new("James", "Gray")
p央视 james.full
puts james.last_first
```
Template Methods

Here’s a feature of ERb that I never see used

---

#85 From Masatoshi Seki
Template Methods

Here’s a feature of ERb that I never see used

require "erb"

class Name
  def initialize(first, last)
    @first = first
    @last  = last
  end
  attr_reader :first, :last
  extend ERB::DefMethod
  def_erb_method("full", "full_name.erb")
  def_erb_method("last_first", "last_name_first.erb")
end

james = Name.new("James", "Gray")
puts james.full
puts james.last_first

James Gray
Gray, James

#85 From Masatoshi Seki
Unix Like File Manipulation

This library has an excellent interface
Unix Like File Manipulation

This library has an excellent interface

#86 From Piotr Szotkowski
require "fileutils"

module FileSystemWork
  extend FileUtils  # or FileUtils::Verbose, FileUtils::DryRun, ...

  module_function

    def do_work
      touch "file.txt"  # or whatever
    end

  end

FileSystemWork.do_work

And You Can Tweak It

Get verbose output or try a dry run before you commit to the real thing

#87 From Eric Hodel
require "fileutils"

module FileSystemWork
  extend FileUtils
  # or FileUtils::Verbose, FileUtils::DryRun, ...

  module_function

    def do_work
      touch "file.txt"  # or whatever
    end
  end
end

FileSystemWork.do_work
require "fileutils"

module FileSystemWork
  extend FileUtils  # or FileUtils::Verbose, FileUtils::DryRun, ...

  module_function

    def do_work
      touch "file.txt"  # or whatever
    end

end

FileSystemWork.do_work

And You Can Tweak It

Get verbose output or try a dry run before you commit to the real thing

#87 From Eric Hodel
require "fileutils"

module FileSystemWork
  extend FileUtils  # or FileUtils::Verbose, FileUtils::DryRun, ...

  module_function

    def do_work
      touch "file.txt"  # or whatever
    end

end

FileSystemWork.do_work

And You Can Tweak It

Get verbose output or try a dry run before you commit to the real thing
require "pathname"

# build paths
dir = Pathname.pwd       # or Pathname.new(...)  
path = dir + __FILE__    # add paths

# work with paths
puts path.realpath
puts path.relative_path_from(dir)
puts

# use paths to do work
path.open do |io|
  5.times do
    puts io.gets
  end
end
require "pathname"

# build paths
dir = Pathname.pwd # or Pathname.new(...)  
path = dir + __FILE__ # add paths

# work with paths
puts path.realpath
puts path.relative_path_from(dir)
puts

# use paths to do work
path.open do |io|
  5.times do
    puts io.gets
  end
end
require "pathname"

# build paths
dir  = Pathname.pwd      # or Pathname.new(...)  
path = dir + __FILE__    # add paths

# work with paths
puts path.realpath       
puts path.relative_path_from(dir)
puts

# use paths to do work
path.open do |io|
  5.times do
    puts io.puts
  end
end

An OO File Interface
Pathname combines File, Dir, File::Stat, and more in one pretty interface

#88
An OO File Interface

Pathname combines File, Dir, File::Stat, and more in one pretty interface
An OO File Interface

Pathname combines File, Dir, File::Stat, and more in one pretty interface
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: \%p" % db["james"]
  puts "Dana: \%p" % db["dana"]
end

The World’s Easiest Database

It’s multiprocessing safe too

#89 From Piotr Szotkowski
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

The World’s Easiest Database

It’s multiprocessing safe too

#89 From Piotr Szotkowski
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

The World’s Easiest Database
It’s multiprocessing safe too

#89 From Piotr Szotkowski
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: \p" % db["james"]
  puts "Dana: \p" % db["dana"]
end

The World’s Easiest Database

It’s multiprocessing safe too
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p " % db["james"]
  puts "Dana: %p " % db["dana"]
end

The World’s Easiest Database

It’s multiprocessing safe too

#89 From Piotr Szotkowski
require "pstore"

db = PStore.new("accounts.pstore")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

The World’s Easiest Database

It’s multiprocessing safe too
require "yaml/store"

db = YAML::Store.new("accounts.yml")

db.transaction do
  db["james"] = 100.00
  db["dana"]  = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

Or Use the Drop-in YAML Version

Same thing, but human readable
require "yaml/store"

db = YAML::Store.new("accounts.yml")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

Or Use the Drop-in YAML Version

Same thing, but human readable

#90 From Piotr Szotkowski
require "yaml/store"

db = YAML::Store.new("accounts.yml")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

---

Or Use the Drop-in YAML Version

Same thing, but human readable

---

#90 From Piotr Szotkowski
require "yaml/store"

db = YAML::Store.new("accounts.yml")

db.transaction do
  db["james"] = 100.00
  db["dana"] = 100.00
end

db.transaction do
  db["dana"] += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.0
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

James: 100.0
Dana: 100.0

Or Use the Drop-in YAML Version

Same thing, but human readable
require "yaml/store"

db = YAML::Store.new("accounts.yml")

db.transaction do
  db["james"] = 100.00
  db["dana"]  = 100.00
end

db.transaction do
  db["dana"]  += 200.00
  db["james"] -= 200.00
  db.abort if db["james"] < 0.
end

db.transaction(true) do
  puts "James: %p" % db["james"]
  puts "Dana: %p" % db["dana"]
end

James: 100.0
Dana: 100.0

---
james: 100.0
dana: 100.0

Or Use the Drop-in YAML Version

Same thing, but human readable

#90 From Piotr Szotkowski
A More Complete Set

Array is nice, but this is safer and more full-featured

#91 From Piotr Szotkowski
A More Complete Set

Array is nice, but this is safer and more full-featured

#91 From Piotr Szotkowski
A More Complete Set

Array is nice, but this is safer and more full-featured

---

#91 From Piotr Szotkowski
A More Complete Set

Array is nice, but this is safer and more full-featured
require "set"

animals = Set.new
animals << "cat"
animals.add("bat")
p animals.add?("rat")
p animals.add?("rat")
p animals

p animals.member?("cat")
p animals.subset?(Set["bat", "cat"])
p animals.superset?(%w[lions tigers bears].to_set)

ordered = SortedSet.new
(1..10).to_a.shuffle.each do |n|
  ordered << n
end
p ordered

A More Complete Set

Array is nice, but this is safer and more full-featured

#91 From Piotr Szotkowski
Tools
Mini Ruby Programs

The command-line switches -n, -p, -i, and -e can save you a lot of work
Mini Ruby Programs

The command-line switches -n, -p, -i, and -e can save you a lot of work

#92 From Benjamin Fleischer
Mini Ruby Programs

The command-line switches -n, -p, -i, and -e can save you a lot of work

#92 From Benjamin Fleischer
The IO-like ARGF

This object wraps Unix style file arguments in an IO interface.
The IO-like ARGF

This object wraps Unix style file arguments in an IO interface
The IO-like ARGF

This object wraps Unix style file arguments in an IO interface

```
ARGF.each_line do |line|
  puts "#{ARGF.lineno}: #{line}"
end
```

```
$ ruby the_iolike_argf.rb one.txt two.txt three.txt
1:  one
2:  two
3:  three
```
You can still use the ARGF magic without Ruby setting it up

```ruby
concatenated_files = ARGF.class.new("one.txt", "two.txt", "three.txt")
concatenated_files.each_line do |line|
  puts line
end
```
You can still use the ARGF magic without Ruby setting it up

#94 From Greg Brown
ARGF Sans the Command-line

You can still use the ARGF magic without Ruby setting it up
The "Flip-flop" Operator

Obscure, but it can do a lot of work for you

#95 From Benjamin Fleischer
The “Flip-flop” Operator

Obscure, but it can do a lot of work for you

#95 From Benjamin Fleischer
Out of Order Code

This Perlish syntax is handy in one-liners

#96 From Nick Seiger
From Nick Seiger

Out of Order Code

This Perlish syntax is handy in one-liners

```bash
$ cat blenders.txt
Ninja   $99.99
Vitamix $378.95
Blendtec $399.99

$ ruby -ne 'BEGIN { total = 0 };
  END { puts "%.2f" % total };  
  total += $_[/\$(\d+(?:\.:\d+)?)/, 1].to_f' blenders.txt

$878.93
```
Out of Order Code

This Perlish syntax is handy in one-liners

#96 From Nick Seiger
IRb’s Last Result

This really pays off in a Rails console as well
IRb’s Last Result

This really pays off in a Rails console as well
IRb’s “Job” Management

Pry really wasn’t the first
IRb’s “Job” Management

Pry really wasn’t the first
IRb’s “Job” Management

Pry really wasn’t the first
IRb’s “Job” Management

Pry really wasn’t the first
IRb’s “Job” Management

Pry really wasn’t the first
It’s often handy to be able to drop into an IRb session

Trigger IRb as Needed

```ruby
require "irb"

def my_program_context
  @my_program_context ||= Struct.new(:value).new(40)
end

trap(:INT) do
  IRB.start
  trap(:INT, "EXIT")
end

loop do
  puts "Current value: #{my_program_context.value}"
  sleep 1
end
```
require "irb"

def my_program_context
  @my_program_context ||= Struct.new(:value).new(40)
end

trap(:INT) do
  IRB.start
  trap(:INT, "EXIT")
end

loop do
  puts "Current value:  #{@my_program_context.value}"
  sleep 1
end
Trigger IRb as Needed

It’s often handy to be able to drop into an IRb session

```ruby
require "irb"

def my_program_context
  @my_program_context ||= Struct.new(:value).new(40)
end

trap(:INT) do
  IRB.start
  trap(:INT, "EXIT")
end

loop do
  puts "Current value:  #{my_program_context.value}"
  sleep 1
end
```
Trigger IRb as Needed

It’s often handy to be able to drop into an IRb session

```
require "irb"

def my_program_context
  @my_program_context ||= Struct.new(:value).new(40)
end

trap(:INT) do
  IRB.start
  trap(:INT)
end

loop do
  puts "Current value: #{my_program_context.value}"
  sleep 1
end

$ ruby trigger_irb_as_needed.rb
Current value: 40
Current value: 40
Current value: 40

^C
1.9.3-p194 :001 > my_program_context.value += 2
=> 42
1.9.3-p194 :002 > exit
Current value: 42
Current value: 42
Current value: 42
```
Trigger IRb as Needed

It’s often handy to be able to drop into an IRb session

```ruby
require 'irb'

def my_program_context
  @my_program_context ||= Struct.new(:value).new(40)
end

trap(:INT) do
  IRB.start
  trap(:INT, "EXIT")
end

loop do
  puts "Current value: #{my_program_context.value}"
  sleep 1
end
```

```
$ ruby trigger_irb_as_needed.rb
Current value: 40
Current value: 40
Current value: 40
^C
```

```
1.9.3-p194 :001 > my_program_context.value += 2
 => 42
1.9.3-p194 :002 > exit
Current value: 42
Current value: 42
Current value: 42
```
$ curl http://twitter.com/statuses/user_timeline/JEG2.json
[

Pretty JSON

This tool ships with Ruby 1.9 or later

#100 From Peter Cooper
Pretty JSON

This tool ships with Ruby 1.9 or later

#100 From Peter Cooper
Pretty JSON

This tool ships with Ruby 1.9 or later

#100 From Peter Cooper
Black Magic
Code That Never Crashes!

Ruby is so powerful that it can make code run forever

```
#!/usr/bin/env ruby -w

at_exit do
  if $! and not [SystemExit, Interrupt].include? $!.class
    exec $PROGRAM_NAME
  end
end

loop do
  left, right = Array.new(2) { rand(-10..10) }
  operator = %w[+ - * /].sample
  puts "#{left} #{operator} #{right} = #{left.send(operator, right)}"
end
```
Code That Never Crashes!

Ruby is so powerful that it can make code run forever

```ruby
#!/usr/bin/env ruby -w

at_exit do
  if $! and not [SystemExit, Interrupt].include? $!.class
    exec $PROGRAM_NAME
  end
end

loop do
  left, right = Array.new(2) { rand(-10..10) }
  operator = %w[+ - * /].sample
  puts "#{left} #{operator} #{right} = #{left.send(operator, right)}"
end
```
#!/usr/bin/env ruby -w

```ruby
at_exit do
  if $! and not [SystemExit, Interrupt].include? $!.class
    exec $PROGRAM_NAME
  end
end

loop do
  left, right = Array.new(2) { rand(-10..10) }
  operator   = %w[+ - * /].sample
  puts "#{left} #{operator} #{right} = #{left.send(operator, right)}"
end
```

Code That Never Crashes!

Ruby is so powerful that it can make code run forever

#101
#!usr/bin/env ruby -w

```ruby
at_exit do
  if $! and not [SystemExit, Interrupt].include? $!.class
    exec $PROGRAM_NAME
  end
end

loop do
  left, right = Array.new(2) { rand(-10..10) }
  operator    = %w[+ - * /].sample
  puts "#{left} #{operator} #{right} = #{left.send(operator, right)}"
end
```

**Code That Never Crashes!**

Ruby is so powerful that it can make code run forever

#101
Code That Never Crashes!

Ruby is so powerful that it can make code run forever
These Slides Will Be Online:
https://speakerdeck.com/u/jeg2
Thanks