

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

42

~~10~~ Things You Didn't Know
Ruby Could Do

101

~~42~~

~~10~~ 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.
```

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.

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.
```

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.
```

Ruby Can Read Your Email

Or any other text content with Ruby code in it

```
puts DATA.read  
—END—  
This is data!
```

Storing Data in Your Code

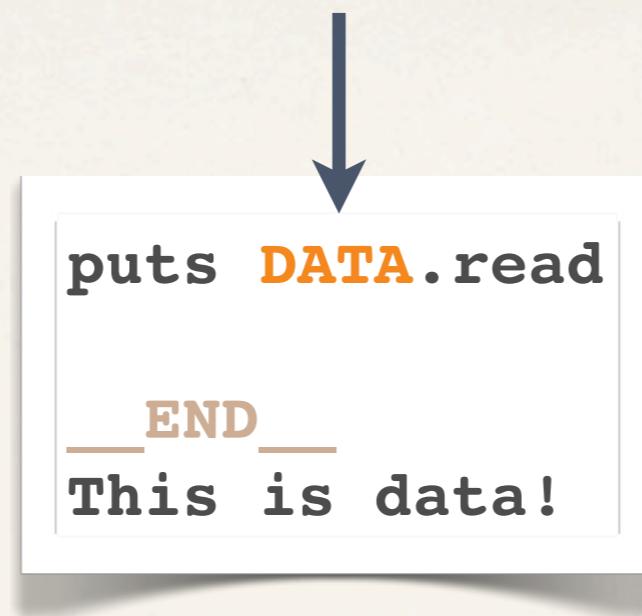
Or is it DATA?

```
puts DATA.read
```

—END— ←
This is data!

Storing Data in Your Code

Or is it DATA?



Storing Data in Your Code

Or is it DATA?

```
puts DATA.read  
  
—END—  
This is data!
```

This is data!

Storing Data in Your Code

Or is it DATA?

```
DATA.rewind  
puts DATA.read  
END
```

A Cheat of a Quine

Most quine rules disallow using IO



A Cheat of a Quine

Most quine rules disallow using IO

```
DATA.rewind
puts DATA.read
__END__
```

```
DATA.rewind
puts DATA.read
__END__
```

A Cheat of a Quine

Most quine rules disallow using IO

```
DATA.flock(File::LOCK_EX | File::LOCK_NB) or abort "Already running."  
  
trap("INT", "EXIT")  
  
puts "Running..."  
loop do  
  sleep  
end  
  
__END__  
DO NOT DELETE: used for locking
```

There Can Be Only One

Use a tricky lock to make your script exclusive

```
DATA.flock(File::LOCK_EX | File::LOCK_NB) or abort "Already running."  
  
trap("INT", "EXIT")  
  
puts "Running..."  
loop do  
  sleep  
end  
  
__END__  
DO NOT DELETE: used for locking
```



There Can Be Only One

Use a tricky lock to make your script exclusive

```
DATA.flock(File::LOCK_EX | File::LOCK_NB) or abort "Already running."  
  
trap("INT", "EXIT")  
  
puts "Running..."  
loop do  
  sleep  
end  
  
—END— ←  
DO NOT DELETE: used for locking
```

There Can Be Only One

Use a tricky lock to make your script exclusive

```
DATA.flock(File::LOCK_EX | File::LOCK_NB) or abort "Already running."  
  
trap("INT", "EXIT")  
  
puts "Running..."  
loop do  
  sleep  
end  
  
END  
DO NOT DELETE:
```

```
$ ruby lock.rb  
Running...  
^Z  
[1]+ Stopped  
$ ruby lock.rb  
Already running.  
$ fg  
ruby lock.rb  
^C$ ruby lock.rb  
Running...
```

There Can Be Only One

Use a tricky lock to make your script exclusive

```
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

```
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

```
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

```
$ ruby -e 'puts { is_this_a_block }' --dump parsetree
#####
## Do NOT use this node dump for any purpose other than ##
## debug and research. Compatibility is not guaranteed. ##
#####

# @ NODE_SCOPE (line: 1)
# +- nd_tbl: (empty)
# +- nd_args:
# |   (null node)
# +- nd_body:
#     @ NODE_ITER (line: 1)
#     +- nd_iter:
#         |   @ NODE_FCALL (line: 1)
#         |   +- nd_mid: :puts
#         |   +- nd_args:
#             |   (null node)
#
...
```

See How Ruby Reads Your Code

All you have to do is ask

```
$ ruby -e 'puts { is_this_a_block }' --dump parsetree
#####
## Do NOT use this node dump for any purpose other than ##
## debug and research. Compatibility is not guaranteed. ##
#####

# @ NODE_SCOPE (line: 1)
# +- nd_tbl: (empty)
# +- nd_args:
# |   (null node)
# +- nd_body:
#     @ NODE_ITER (line: 1)
#     +- nd_iter:
#         |   @ NODE_FCALL (line: 1)
#         |   +- nd_mid: :puts
#         |   +- nd_args:
#             |   (null node)
#
...
```



See How Ruby Reads Your Code

All you have to do is ask

```
$ ruby -e 'puts { is_this_a_block }' --dump parsetree
#####
## Do NOT use this node dump for any purpose other than ##
## debug and research. Compatibility is not guaranteed. ##
#####

# @ NODE_SCOPE (line: 1)
# +- nd_tbl: (empty)
# +- nd_args:
# |   (null node)
# +- nd_body:
#     @ NODE_ITER (line: 1)
#     +- nd_iter:
#         |   @ NODE_FCALL (line: 1)
#         |   +- nd_mid: :puts ←————
#         |   +- nd_args:
#         |       (null node)
#
...
```

See How Ruby Reads Your Code

All you have to do is ask

```
$ ruby -e 'puts { is_this_a_block }' --dump parsetree_with_comment
...
#      @ NODE_ITER (line: 1)
#      | # method call with block
#      | # format: [nd_iter] { [nd_body] }
#      | # example: 3.times { foo }
#      +- nd_iter (iteration receiver):
#          | @ NODE_FCALL (line: 1)
#          | | # function call
#          | | # format: [nd_mid]([nd_args])
#          | | # example: foo(1)
#          | +- nd_mid (method id): :puts
#          | +- nd_args (arguments):
#              | | (null node)
# ...
...
```

Available With Comments

Ruby can even explain it to you

```
$ ruby -e 'puts { is_this_a_block }' --dump parsetree_with_comment  
...  
#      @ NODE_ITER (line: 1)  
#      | # method call with block  
#      | # format: [nd_iter] { [nd_body] }  
#      | # example: 3.times { foo }  
#      +- nd_iter (iteration receiver):  
#          @ NODE_FCALL (line: 1)  
#          | # function call  
#          | # format: [nd_mid]([nd_args])  
#          | # example: foo(1)  
#          +- nd_mid (method id): :puts  
#          +- nd_args (arguments):  
#              (null node)  
...
```



Available With Comments

Ruby can even explain it to you

```
$ ruby -e 'puts { is_this_a_block }' --dump parsetree_with_comment
...
#      @ NODE_ITER (line: 1)
#      | # method call with block ←
#      | # format: [nd_iter] { [nd_body] }
#      | # example: 3.times { foo }
#      +- nd_iter (iteration receiver):
#          | @ NODE_FCALL (line: 1)
#          | | # function call
#          | | # format: [nd_mid]([nd_args])
#          | | # example: foo(1)
#          | +- nd_mid (method id): :puts
#          | +- nd_args (arguments):
#              | | (null node)
# ...
...
```

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
0002 putobject        40
0004 putobject        2
0006 opt_plus          <ic:2>
0008 setdynamic        ft, 0
0011 trace           1
0013 putself
0014 getdynamic        ft, 0
0017 send             :p, 1, nil, 8, <ic:1>
0023 leave
```

View the Machine Instructions

Alternately, you can view the virtual machine instructions

```
$ 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
0002 putobject        40
0004 putobject        2
0006 opt_plus          <ic:2>
0008 setdynamic        ft, 0
0011 trace           1
0013 putself
0014 getdynamic        ft, 0
0017 send             :p, 1, nil, 8, <ic:1>
0023 leave
```

View the Machine Instructions

Alternately, you can view the virtual machine instructions

```
$ 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

```
$ 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

```
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”

```
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”

```
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”

```
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”

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

p factorial(30_000)
```

Tail Call Optimization

Yeah, it's in there

```
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

```
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

```
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

```
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

```
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)
```

```
27595372462193845993799421664254627839807...
```

Tail Call Optimization

Yeah, it's in there

Syntax

```
class Parent
  def show_args(*args)
    p args
  end
end

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

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

Pass Some Arguments Up

Helps add functionality to methods

```
class Parent
  def show_args(*args)
    p args
  end
end

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

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

Pass Some Arguments Up

Helps add functionality to methods

```
class Parent
  def show_args(*args)
    p args
  end
end

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

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

[:a, :b]

Pass Some Arguments Up

Helps add functionality to methods

```
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 }
```

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 }
```

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 }
```

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

Pass The Same Arguments Up

Remember, super is a magical keyword

```
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

```
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

```
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

```
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

```
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

```
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

```
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 }
```

Pass No Block Up

How to make a block disappear

```
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 }
```

Pass No Block Up

How to make a block disappear

```
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

Pass No Block Up

How to make a block disappear

```
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
```

Asking If You Can Pass Up

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

```
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
```

Asking If You Can Pass Up

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

```
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
```

Modified: DelegateToMe
DelegateIfICan

Asking If You Can Pass Up

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

```
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”



```
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”

```
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”

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

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

```
:called
[:arg, :default, #<Proc:0x007fe602887878@.../lambda_literals.rb:5>]
```

Lambda Literals

Also known as the “stabby lambda”

```
# encoding: UTF-8

module Kernel
    alias_method :λ, :lambda
end

l = λ { p :called }
l.call
```

Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us

```
# encoding: UTF-8 ←  
  
module Kernel  
    alias_method :λ, :lambda  
end  
  
l = λ { p :called }  
l.call
```

Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us

```
# encoding: UTF-8

module Kernel
    alias_method :λ, :lambda
end

l = λ { p :called }
l.call
```

Or Make Your Own UTF-8 Syntax

Perl 6 has nothing on us

```
# encoding: UTF-8

module Kernel
    alias_method :λ, :lambda
end

l = λ { p :called }
l.call
```

:called

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

```
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

```
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>]
```

Blocks Can Now Take Blocks

This trick helps with metaprogramming

```
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?

```
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?

```
class Callable
  def call
    :my_own_class
  end
end

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

```
:lambda
Array
:my_own_class
```

The New Call Anything Syntax

Or should we call it the “Call Me Maybe” syntax?

```
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

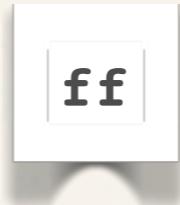
This probably isn't a great thing to abuse, but still... shiny

```
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

```
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

```
p (1..10).inject(:*) # instead of: inject(&:*)
```

inject() a Symbol

This is even shorter than using Symbol#to_proc

```
p (1..10).inject(:*) # instead of: inject(&:*)
```



inject() a Symbol

This is even shorter than using `Symbol#to_proc`

```
p (1..10).inject(:*) # instead of: inject(&:*)
```

```
3628800
```

inject() a Symbol

This is even shorter than using Symbol#to_proc

```
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

```
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

```
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 Ranges

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

```
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

```
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

```
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

```
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 ===

```
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 ===

```
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

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

A Formatted Output Syntax

Like sprintf(), but even shorter

```
def debug(name, content)
  puts "%s: %p" % [name, content] ←
end

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

A Formatted Output Syntax

Like sprintf(), but even shorter

```
def debug(name, content)
  puts "%s: %p" % [name, content]
end

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

A Formatted Output Syntax

Like sprintf(), but even shorter

```
def debug(name, content)
  puts "%s: %p" % [name, content]
end

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

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

A Formatted Output Syntax

Like sprintf(), but even shorter

```

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

```
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

```

Item	Price

Item 1	\$10.00
Item 2	\$19.99
Item 3	\$ 4.50

Or By Name

A Ruby 1.9 enhancement to these format patterns

```
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

```
$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

```
$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

```
$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

```
$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

```
$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

```
$VERBOSE = true

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

p WarnMe.new.var
```

42

Dodging a Warning

The “or equals” operator includes a defined? check

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

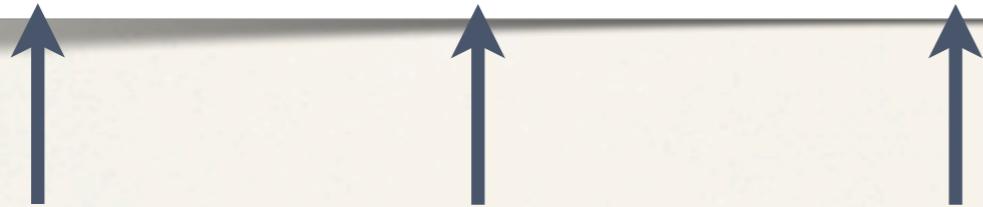
puts "#@instance, #@@class, and #\$global variables don't need braces."
```

Shortcut Variable Interpolation

The braces are optional with a sigil toting variable

```
@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

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

puts "#@instance, #@@class, and #\$global variables don't need braces."
```

instance, class, and global variables don't need braces.

Shortcut Variable Interpolation

The braces are optional with a sigil toting variable

```
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
```

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

Variables From a Regex

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

```
[ ["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

```
[ ["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

```
[ ["James", "Gray", 36],  
  ["Dana", "Gray", 37],  
  ["Summer", "Gray", 2] ].each do |name, ignore, ignore|  
  puts name  
end
```

```
/.../the_unused_variable.rb:3: duplicated argument name
```

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, _, _|  
  puts name  
end
```

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, _, _|  
  puts name  
end
```



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, _, _|  
  puts name  
end
```

James
Dana
Summer

The Unused Variable

Ruby supports the convention of `_` as an unused variable name

Data Structures

```
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

```
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

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

position = ring
4.times do
  puts position.first
  position = position.last
end
```

one
two
three
one

Objects Can Contain Themselves

Array Inception

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

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

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



Or Just Use `cycle()`

Ruby 1.9 has a nice new repeating iterator

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

one
two
three
one

Or Just Use cycle()

Ruby 1.9 has a nice new repeating iterator

```
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

```
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

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

```
[ "James", "Gray" ]  
[ "Yukihiro", "Matsumoto" ]
```

Associative Arrays

An ordered (by your criteria) Hash-like thing

```
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

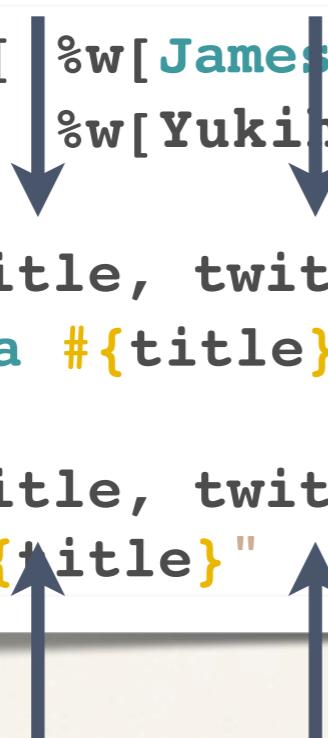
Sneak some extra data through on the end

```
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

```
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

```
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}"
```

matz is a Language Designer
I am a Developer

More Than Two Fields

Sneak some extra data through on the end

```
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

```
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

```
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

```
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}."
```

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

```
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

```
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

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

p fibonacci[300]
```

```
222232244629420445529739893461909967206666939096499764990979600
```

Easy, Fast Memoization

Ruby's Hash is a memoization algorithm in disguise

```
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

```
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

```
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

```
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

```

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

```

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

```

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

```

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)]
end

```

```

$ ruby rpn.rb
>> 2
0: 2
>> 3
0: 2
1: 3
>> *
0: 6
>> 2
0: 6
1: 2
>> /
0: 3
>> n
0: -3

```

Rolling Your Own Dispatch Table

Works great with the new Hash and lambda syntax

```

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

```

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{[-*/]\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

```

class Input;
class Number < Input; def calculate() @input.to_n.send("#@")
class UnaryOperation < Input; def calculate(n) n.send("#@")
class BinaryOperation < Input; def calculate(l, r) l.send(@in

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

```

```

$ ruby rpn.rb
>> 2
0: 2
>> 3
0: 2
1: 3
>> *
0: 6
>> 2
0: 6
1: 2
>> /
0: 3
>> n
0: -3

```

Or Just Use Ruby to Dispatch

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

```
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

```
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

```
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

```
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

```
params = {var: 42}

p params.fetch(:var)

p params.fetch(:missing, 42)
p params.fetch(:missing) { 40 + 2 }

params.fetch(:missing)
```

```
42
42
42
/usr/local/bundle/gems/activesupport-5.2.4/lib/active_support/core_ext/hash/keys.rb:29:in `fetch': key not found: :missing (KeyError)
from /usr/local/bundle/gems/activesupport-5.2.4/lib/active_support/core_ext/hash/keys.rb:29:in `block in fetch'
```

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

```
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

```
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

```
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

I use this more often than the match operator

```
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

```
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

```
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

```
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

```
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

```
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

```
expression = "40 + 2 = 42"  
  
p expression[expression.rindex(/\d+/)..-1]  
p expression[expression.rindex(/\b\d+/)..-1]
```

"2"
"42"

Finding the Last Match

Work from the right instead of the left

```
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

```
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

```
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
```

Chew Through Binary Data

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

Iterators

```
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

```
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

```
letters = "a".."d"  
numbers = 1..3  
  
letters.zip(numbers) 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}
```

Iterating in Lockstep

This can walk through two or more collections at once

```
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}" }
```

Partition Your Data

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

```
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}" }
```

Partition Your Data

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

```
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

```
headings = [ "1.1 Compiler Tricks",
             "1.2 Syntax",
             "1.3 Data Structures",
             "1.4 Iterators",
             "2.1 Core Ruby",
             "2.2 The Standard Library",
             "2.3 Tools",
             "2.4 Black Magic" ]
headings.chunk { |heading| heading[/^A\d+/] }
               .each do |chapter, headings|
                 puts "Chapter #{chapter}:"
                 puts headings.map { |heading| "  #{heading}" }
   end
```

Take Data in Chunks

A newer iterator for us to exploit

```
headings = [ "1.1 Compiler Tricks",
             "1.2 Syntax",
             "1.3 Data Structures",
             "1.4 Iterators",
             "2.1 Core Ruby",
             "2.2 The Standard Library",
             "2.3 Tools",
             "2.4 Black Magic" ]
headings.chunk { |heading| heading[/\A\d+/] } ←
  .each do |chapter, headings|
    puts "Chapter #{chapter}:"
    puts headings.map { |heading| "  #{heading}" }
end
```

Take Data in Chunks

A newer iterator for us to exploit

```
headings = [ "1.1 Compiler Tricks",
             "1.2 Syntax",
             "1.3 Data Structures",
             "1.4 Iterators",
             "2.1 Core Ruby",
             "2.2 The Standard Library",
             "2.3 Tools",
             "2.4 Black Magic" ]
headings.chunk { |heading| heading[/\d/]}
               .each do |chapter, headings|
                 puts "Chapter #{chapter}:"
```

puts headings.map { |heading| " #{"

```
end
```

Chapter 1:

```
1.1 Compiler Tricks
1.2 Syntax
1.3 Data Structures
1.4 Iterators
```

Chapter 2:

```
2.1 Core Ruby
2.2 The Standard Library
2.3 Tools
2.4 Black Magic
```

Take Data in Chunks

A newer iterator for us to exploit

```
require "pp"

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

pp chess_squares
```

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
```



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
```

```
[ "A1",
  "A2",
  "A3",
  "A4",
  "A5",
  "A6",
  "A7",
  "A8",
  "B1",
  "B2",
  "B3",
  "B4",
  "B5",
  "B6",
  "B7",
  "B8",
  "C1",
  ...]
```

map() + flatten() = flat_map()

This can be a handy shortcut for working with nested collections

```
# 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()

```
# 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()

```
# 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()

```
# 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}
```

Replace Ugly inject() Calls

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

```
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

```
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

```
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]
```

Take a Little Off the Top

Pull or skip from the beginning of a list

```
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

```
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

```
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

```
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>"
```

```
<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>
```

Manual Iteration

You can step through the new Enumerator objects by hand

```
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: #{$.!.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: #{$.!.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: #{$.!.class}"

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

```
Error raised: StopIteration
Processing cat...
Processing bat...
Processing rat...
```

A Smarter loop()

Or get some help from loop()

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

Chaining Iterators

Enumerators also allow for the chaining of iteration

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

Chaining Iterators

Enumerators also allow for the chaining of iteration

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

```
[ "abc", "cde", "def", "efg", "ghi", "hij", "ijk", ... ]
```

Chaining Iterators

Enumerators also allow for the chaining of iteration

```

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

```

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

```

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

```

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 }

Add an Index to Any Iterator

Enumerator supports this special method just for chaining

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

```

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

```

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

```

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

```

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

```

42
:called

Programmatically Build Modules

Modules can also be constructed by code

```

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

```

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

```

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

Inherit From an Expression

Programmatically build parent classes

```

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

```

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

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
    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

```

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

```

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
  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

```

class LimitedUse < Module
  def initialize(limit)
    @limit = limit
  end
  super() do
    define_singleton_method(:included) do |parent|
      count = 0
      parent.send(:define_method, :to_s) do
        count += 1
        if count >= limit
          raise RuntimeError, "Over use limit"
        end
        count.to_s
      end
    end
  end
end

[ShortLived, LimitedUse.new(3), Object, Kernel, BasicObject]
#<ShortLived:0x007fe1f3084558>
#<ShortLived:0x007fe1f3084558>
#<ShortLived:0x007fe1f3084558>
#<ShortLived:0x007fe1f3084558>
/.../subclass_module.rb:9:in `block (4 levels) in initialize':
  Over use limit (RuntimeError)
from /.../subclass_module.rb:22:in `<main>'
```

```

  def to_s; "#<#{self.class.name}:#{self.object_id}>" 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

```
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
```

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
```

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]
```

Empty Types

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

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

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

Don't Inherit From Struct

Struct takes a block

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

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

Don't Inherit From Struct

Struct takes a block

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

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

James Gray

Don't Inherit From Struct

Struct takes a block

```
# 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

```
# 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

```
# 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

```
# 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
```

James Gray

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

```

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

```

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

```

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

```

true
Aloha James Gray

Be a Good `method_missing()` User

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

```

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

```

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

```

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

```

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

```

42
:other

instance_eval() In Disguise

This trick is faster than the real thing

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

Iterating Over Each Object

Remember, we're talking about MRI here

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



Iterating Over Each Object

Remember, we're talking about MRI here

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

```
.ext
  CONFIG[ "PREP" ] = "miniruby$(EXEEXT)"
EXTOUT
no
LIBRUBY_RELATIVE

ARCHFILE

EXECUTABLE_EXTS
nodoc
...
```

Iterating Over Each Object

Remember, we're talking about MRI here

```
ObjectSpace.each_object(String) do |object|
  puts object
end
```

Iterating Over Specific Types

Focus in on specific objects

```
ObjectSpace.each_object(String) do |object|
  puts object
end
```



Iterating Over Specific Types

Focus in on specific objects

```
ObjectSpace.each_object(String) do |object|
  puts object
end
```

```
.ext
  CONFIG[ "PREP" ] = "miniruby$(EXEEXT)"
EXTOUT
no
LIBRUBY_RELATIVE

ARCHFILE

EXECUTABLE_EXTS
nodoc
...
```

Iterating Over Specific Types

Focus in on specific objects

```
require "pp"  
pp ObjectSpace.count_objects
```

Count All Objects

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

```
require "pp"  
pp ObjectSpace.count_objects
```



Count All Objects

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

```
require "pp"
pp ObjectSpace.count_obj
```

```
{ :TOTAL=>17579,
  :FREE=>98,
  :T_OBJECT=>9,
  :T_CLASS=>496,
  :T_MODULE=>23,
  :T_FLOAT=>7,
  :T_STRING=>6466,
  :T_REGEXP=>28,
  :T_ARRAY=>1222,
  :T_HASH=>15,
  :T_BIGNUM=>6,
  :T_FILE=>10,
  :T_DATA=>557,
  :T_MATCH=>109,
  :T_COMPLEX=>1,
  :T_NODE=>8510,
  :T_ICLASS=>22}
```

Count All Objects

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

```
GC::Profiler.enable

10.times do
  array = Array.new(1_000_000) { |i| i.to_s }
end

puts GC::Profiler.result
```

Profile the Garbage Collector

This can help find memory leaks

```
GC::Profiler.enable ←————  
  
10.times do  
  array = Array.new(1_000_000) { |i| i.to_s }  
end  
  
puts GC::Profiler.result
```

Profile the Garbage Collector

This can help find memory leaks

```
GC::Profiler.enable  
  
10.times do  
  array = Array.new(1_000_000) { |i| i.to_s }  
end  
  
puts GC::Profiler.result ←
```

Profile the Garbage Collector

This can help find memory leaks

GC::Profiler.enable

GC 21 invokes.

Index	Invoke Time(sec)	Use Size(byte)	Total Size(byte)	Total Object	GC Time(ms)
1	0.010	305000	703480	17587	0.52899999999999991473
2	0.013	588840	719840	17996	0.304000000000000043654
3	0.016	1258720	1276080	31902	0.5030000000000000266
4	0.022	2256520	2274040	56851	0.77800000000000091305
5	0.032	4055240	4073640	101841	1.40099999999999935696
6	0.050	7292960	7312920	182823	2.639000000000000245493
7	0.084	13114280	13137080	328427	4.41699999999999004530
8	0.144	23595920	23623840	590596	7.70400000000001661249
9	0.253	2466120	42486920	1062173	1.61999999999995480948
10	0.410	30496440	42486920	1062173	13.2160000000000463274
11	0.423	30496480	30544120	763603	10.4390000000003203127
12	0.568	14909960	54936880	1373422	7.1509999999990743049
13	0.770	11018240	54936880	1373422	3.7270000000003583978
14	0.932	40182760	46773240	1169331	22.61799999999991683808
15	1.169	40182800	80229440	2005736	26.7779999999997027089
16	1.398	40182840	80229440	2005736	27.8210000000009742962
17	1.628	40182840	80229440	2005736	27.3839999999985112709
18	1.858	40182840	80229440	2005736	27.5300000000005442757
19	2.089	40182840	80229440	2005736	27.8369999999988961008
20	2.322	40182760	80229440	2005736	27.5969999999964958874

Profile the Garbage Collector

This can help find memory leaks

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

Tapping Into a Call Chain

Also used to mutate without affecting the return value

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

James

Tapping Into a Call Chain

Also used to mutate without affecting the return value

```
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

```
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

```
puts "sssssssssssemaj".sub(/\A(\w)\1+/, '\1')
  .tap { |str| p str.size }
  .reverse
  .capitalize
```

5
James

Tapping Into a Call Chain

Also used to mutate without affecting the return value

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

Sneaking in a p() Without tap()

In Ruby 1.9, p() returns its argument

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



Sneaking in a p() Without tap()

In Ruby 1.9, p() returns its argument

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

"semaj"
James

Sneaking in a p() Without tap()

In Ruby 1.9, p() returns its argument

```
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

```
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

```
Thread.abort_on_exception = true

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

loop do
  sleep
end
```

```
.../bubbling_up_thread_errors.rb:4:in `block in <main>':
Oops, we can't continue (RuntimeError)
```

Bubbling Up Thread Errors

This can be quite helpful when debugging threaded code

```
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

```
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

```
def var
  @var || 40
end
```

```
$ ruby the_debug_flag.rb
42
$ ruby -d the_debug_flag.rb
Exception `LoadError' at /.../rubygems.rb:1264 -
  cannot load such file -- rubygems/defaults/operating_system
Exception `LoadError' at /.../rubygems.rb:1273 -
  cannot load such file -- rubygems/defaults/ruby
the_debug_flag.rb:2: warning: instance variable @var not initialized
var is 40
the_debug_flag.rb:2: warning: instance variable @var not initialized
42
```

The \$DEBUG Flag

Turn your debugging code on and off as needed

```
def var
  @var || 40
end
```

```
$ ruby the_debug_flag.rb
42
$ ruby -d the_debug_flag.rb
Exception `LoadError' at /.../rubygems.rb:1264 -
  cannot load such file -- rubygems/defaults/operating_system
Exception `LoadError' at /.../rubygems.rb:1273 -
  cannot load such file -- rubygems/defaults/ruby
the_debug_flag.rb:2: warning: instance variable @var not initialized
var is 40 ←
the_debug_flag.rb:2: warning: instance variable @var not initialized
42
```

The \$DEBUG Flag

Turn your debugging code on and off as needed

```
p 2.between?(1, 10)  
p "cat".between?("bat", "rat")
```

Two Tests in One

This works on any Comparable object

```
p 2.between?(1, 10)  
p "cat".between?("bat", "rat")
```

Two Tests in One

This works on any Comparable object

```
p 2.between?(1, 10)  
p "cat".between?("bat", "rat")
```

true
false

Two Tests in One

This works on any Comparable object

```
File.foreach(__FILE__) do |line|
  p line
end
```

Line-by-line Reading

foreach() = open() + each()

```
File.foreach(__FILE__) do |line|  
  p line  
end
```



Line-by-line Reading

foreach() = open() + each()

```
File.foreach(__FILE__) do |line|
  p line
end
```

```
"File.foreach(__FILE__) do |line|\\n"
"  p line\\n"
"end\\n"
```

Line-by-line Reading

foreach() = open() + each()

```
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()

```
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()

```
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()

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

one
two
three

one
one and a half
two

one
two
three

The Write-a-file Shortcut

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

```
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

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
```

Daemonize Your Process

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

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

```
#!/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()`

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

```
#!/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()`

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

```
#!/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()`

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

```
#!/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()`

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

```
#!/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()`

```

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

#!/usr/bin/ruby
puts "SOME_VAR=42"
$stdin.read:
pid = spawn( {"SOME_VAR" => "42"}, # env
             "./child_process.rb", # the command
             in: open(__FILE__) ) # options
Process.wait(pid)

puts "$stdin.read:", $stdin.read

```

Process Launching on Steroids

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

```

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", "*.*")                      # bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.*")          # 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])          # open IO
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

```

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", "*.*")                      # bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.*")          # 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])          # open IO
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

```

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", "*.*")                       #bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.*")           # 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])          # open IO
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

```

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", "*.*")                       # bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.*")           # 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])          # open IO
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

```

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", "*.*")                      # bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.*")           # 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

```

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", "*.*")                      # bypass the shell, no expansion
spawn(["rm", "sweeper"], "*.*")           # 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])          # open IO
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

The Standard Library

```
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

```
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

```
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

```
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

```
require "securerandom"  
  
p SecureRandom.random_number  
p SecureRandom.random_number(100)
```

```
0.83375534446421
```

```
50
```

```
"5f1edc66708381c18f5527ccc49c0fad7822d5f3"  
"aoRvYu/V6/MeluooKG+tf4yjyyU="  
"WfWpdByTVMd1iaymKB_FmqTGgE8"  
" !d\x94\xE6viS\xCB\xA7\xDB\x84\xCF\x8EY\xCBI\x9F6\xE6\xFD"  
  
"5243d395-f317-4394-aa5d-762005d1fe7a"
```

Get Random Data

Ruby can generate random data for you

```
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

```
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

```
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

```
require "open-uri"

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

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

```
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'])
```



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'])
```

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'])
```



Escape For the Shell

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

```
require "shellwords"

p Shellwords.shellwords('one "two" three')           ["one", "two", "a longer three"]
p Shellwords.shellwords("one 'two' three")            ["one", "two", "a longer three"]
p Shellwords.shellwords('escaped \\"quote\\" characters') ["escaped \"quote\" characters"]
p Shellwords.shellwords('escaped spaces')              ["escaped spaces"]
p Shellwords.shellwords(%Q{'back to back quoting'})  ["back to back quoting"]
p Shellwords.shellescape("two words")                 "two\\ words"
p Shellwords.shellescape('"quotes" included')         "\"\\\"\"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 "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

```
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

```
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

```
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
```

<%= first %> <%= last %>

<%= last %>, <%= first %>

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
```

<%= first %> <%= last %>

<%= last %>, <%= first %>

James Gray
Gray, James

Template Methods

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

```
require "fileutils"

FileUtils.mkdir    dir, options
FileUtils.mkdir_p  dir, options
FileUtils.ln       src, dest, options
FileUtils.ln_s    src, dest, options
FileUtils.cp       src, dest, options
FileUtils.cp_r   src, dest, options
FileUtils.mv      src, dest, options
FileUtils.rm      files, options
FileUtils.rm_r   files, options
FileUtils.rm_rf  files, options
FileUtils.chmod   mode, files, options
FileUtils.chmod_R mode, files, options
FileUtils.chown   user, group, files, options
FileUtils.chown_R user, group, files, options
FileUtils.touch  files, options
```

Unix Like File Manipulation

This library has an excellent interface

```
require "fileutils"

FileUtils.mkdir    dir, options
FileUtils.mkdir_p  dir, options
FileUtils.ln       src, dest, options
FileUtils.ln_s    src, dest, options
FileUtils.cp       src, dest, options
FileUtils.cp_r   src, dest, options
FileUtils.mv      src, dest, options
FileUtils.rm      files, options
FileUtils.rm_r   files, options
FileUtils.rm_rf  files, options
FileUtils.chmod   mode, files, options
FileUtils.chmod_R mode, files, options
FileUtils.chown   user, group, files, options
FileUtils.chown_R user, group, files, options
FileUtils.touch  files, options
```

Unix Like File Manipulation

This library has an excellent interface

```
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 "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 "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 "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
```

An OO File Interface

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

```
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
```

An OO File Interface

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

```
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
```

An OO File Interface

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

```
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
```

An OO File Interface

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

```
require "pathname"

# build paths
dir = Pathname.pwd      # or Pathname.new(...)

/Users/james/Desktop/the_standard_library/an_oo_file_interface.rb
an_oo_file_interface.rb

require "pathname"

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

5.times do
  puts io.gets
end
end
```

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

```
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

```
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

```
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

```

James: 100.0
Dana: 100.0

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

```
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

```

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

```
require "set"

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

p animals.member?("tiger")
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

```
require "set"

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

p animals.member?("tiger")
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

```

require "set"

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

p animals.member?("tiger")
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

```

require "set"

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

p animals.add?("rat")



p animals.add?("rat")



p animals



p animals.member?("tiger")



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

```

require "set"

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

p animals.add?("rat")



p animals.add?("rat")



p animals



p animals.member?("rat")



p animals.subset?(Set.new(1..10))



p animals.superset?(Set.new(1..10))



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

p ordered


```

```

#<Set: {"cat", "bat", "rat"}>
nil
#<Set: {"cat", "bat", "rat"}>
false
false
false
#<SortedSet: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}>

```

A More Complete Set

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

Tools

```
$ cat data.txt
ones
twos
threes
$ ruby -pi.bak -e 'sub(/s\z/, "")' data.txt
$ cat data.txt
one
two
three
$ cat data.txt.bak
ones
twos
threes
```

Mini Ruby Programs

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

```
$ cat data.txt  
ones  
twos  
threes  
$ ruby -pi.bak -e 'sub(/s\z/, "")' data.txt  
$ cat data.txt  
one  
two  
three  
$ cat data.txt.bak  
ones  
twos  
threes
```



```
ARGF.each_line do |$_|  
  # -e code goes here  
  print $_ # for -p, not -n  
end
```

Mini Ruby Programs

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

```
$ cat data.txt  
ones  
twos  
threes  
$ ruby -pi.bak -e 'sub(/s\z/, "")' data.txt  
$ cat data.txt  
one  
two  
three  
$ cat data.txt.bak ←  
ones  
twos  
threes
```



Mini Ruby Programs

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

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

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
```

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
```

The IO-like ARGF

This object wraps Unix style file arguments in an IO interface

```
concatenated_files = ARGF.class.new("one.txt", "two.txt", "three.txt")
concatenated_files.each_line do |line|
  puts line
end
```

ARGF Sans the Command-line

You can still use the ARGF magic without Ruby setting it up

```
concatenated_files = ARGF.class.new("one.txt", "two.txt", "three.txt")
concatenated_files.each_line do |line|
  puts line
end
```

ARGF Sans the Command-line

You can still use the ARGF magic without Ruby setting it up

```
concatenated_files = ARGF.class.new("one.txt", "two.txt", "three.txt")
concatenated_files.each_line do |line|
  puts line
end
```

one
two
three

ARGF Sans the Command-line

You can still use the ARGF magic without Ruby setting it up

```
$ cat numbers.txt
one
two
three
four
five
six
seven
eight
nine
ten
$ ruby -ne 'print if /\At/../e\z/' numbers.txt
two
three
ten
```

The “Flip-flop” Operator

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

```
$ cat numbers.txt
one
two
three
four
five
six
seven
eight
nine
ten
$ ruby -ne 'print if /\At/../e\z/' numbers.txt
two
three
ten
```



The “Flip-flop” Operator

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

```
$ 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

```
$ 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

```
$ 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

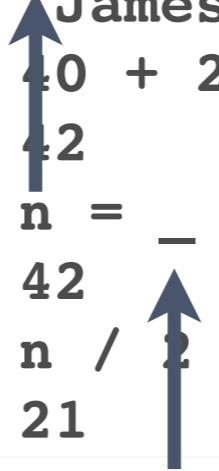
This Perlish syntax is handy in one-liners

```
>> "semaj".reverse  
=> "james"  
>> _.capitalize  
=> "James"  
>> 40 + 2  
=> 42  
>> n = -  
=> 42  
>> n / 2  
=> 21
```

IRb's Last Result

This really pays off in a Rails console as well

```
>> "semaj".reverse  
=> "james"  
>> _.capitalize  
=> "James"  
>> 40 + 2  
=> 42  
>> n = -  
=> 42  
>> n / ?  
=> 21
```



IRb's Last Result

This really pays off in a Rails console as well

```
>> self
=> main
>> irb "james"
>> capitalize
=> "James"
>> jobs
=> #0->irb on main (#<Thread:0x007ff80406b5b8>: stop)
#1->irb#1 on james (#<Thread:0x007ff805856da8>: running)
>> fg 0
=> #<IRB::Irb: ...>
>> self
=> main
>> fg 1
=> #<IRB::Irb: ...>
>> self
=> "james"
>> exit
=> #<IRB::Irb: ...>
>> self
=> main
>> exit
```

IRb’s “Job” Management

Pry really wasn’t the first

```
>> self
=> main
>> irb "james" ←
>> capitalize
=> "James"
>> jobs
=> #0->irb on main (#<Thread:0x007ff80406b5b8>: stop)
#1->irb#1 on james (#<Thread:0x007ff805856da8>: running)
>> fg 0
=> #<IRB::Irb: ...>
>> self
=> main
>> fg 1
=> #<IRB::Irb: ...>
>> self
=> "james"
>> exit
=> #<IRB::Irb: ...>
>> self
=> main
>> exit
```

IRb’s “Job” Management

Pry really wasn’t the first

```
>> self
=> main
>> irb "james"
>> capitalize
=> "James"
>> jobs ←————
=> #0->irb on main (#<Thread:0x007ff80406b5b8>: stop)
#1->irb#1 on james (#<Thread:0x007ff805856da8>: running)
>> fg 0
=> #<IRB::Irb: ...>
>> self
=> main
>> fg 1
=> #<IRB::Irb: ...>
>> self
=> "james"
>> exit
=> #<IRB::Irb: ...>
>> self
=> main
>> exit
```

IRb’s “Job” Management

Pry really wasn’t the first

```
>> self
=> main
>> irb "james"
>> capitalize
=> "James"
>> jobs
=> #0->irb on main (#<Thread:0x007ff80406b5b8>: stop)
#1->irb#1 on james (#<Thread:0x007ff805856da8>: running)
>> fg 0 ←←←
=> #<IRB::Irb: ...>
>> self
=> main
>> fg 1 ←←←
=> #<IRB::Irb: ...>
>> self
=> "james"
>> exit
=> #<IRB::Irb: ...>
>> self
=> main
>> exit
```

IRb’s “Job” Management

Pry really wasn’t the first

```
>> self
=> main
>> irb "james"
>> capitalize
=> "James"
>> jobs
=> #0->irb on main (#<Thread:0x007ff80406b5b8>: stop)
#1->irb#1 on james (#<Thread:0x007ff805856da8>: running)
>> fg 0
=> #<IRB::Irb: ...>
>> self
=> main
>> fg 1
=> #<IRB::Irb: ...>
>> self
=> "james"
>> exit ←————
=> #<IRB::Irb: ...>
>> self
=> main
>> exit
```

IRb’s “Job” Management

Pry really wasn’t the first

```
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, "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, "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
  @my_program = 40
end

trap(:INT) do
  IRB.start
  trap(:INT, &method(:exit))
end

loop do
  puts "Current value: #{@my_program}"
  sleep 1
end
```

```
$ ruby trigger_irb_as_needed.rb
Current value: 40
Current value: 40
Current value: 40
^C1.9.3-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

```
require "irb"

def my_program
  @my_program = 40
end

trap(:INT) do
  IRB.start
  trap(:INT, &method(:exit))
end

loop do
  puts "Current value: #{@my_program}"
  sleep 1
end
```

```
$ ruby trigger_irb_as_needed.rb
Current value: 40
Current value: 40
Current value: 40
^C1.9.3-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

```
$ curl http://twitter.com/statuses/user_timeline/JEG2.json
[{"id_str":"253982951187034113","place":null,"retweeted":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_screen_name":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"user":{"id":20941662,"profile_image_url":"http://a0.twimg.com/profile_images/2311650093/fkgorpzafxmsafxp6wi_normal.png","profile_background_image_url_https":"https://si0.twimg.com/images/themes/theme7/bg.gif","location":"Edmond, OK","profile_use_background_image":true,"profile_text_color":"333333","follow_request_sent":false,"default_profile":false,...}}
```

Pretty JSON

This tool ships with Ruby 1.9 or later

```
$ curl http://twitter.com/statuses/user_timeline/JEG2.json | prettify_json.rb
[
{
  "id_str": "253982951187034113",
  "place": null,
  "retweeted": false,
  "in_reply_to_status_id_str": null,
  "in_reply_to_status_id": null,
  "in_reply_to_screen_name": null,
  "in_reply_to_user_id_str": null,
  "user": {
    "id": 20941662,
    "profile_image_url": "http://.../fkgorpzafxmsafxp6wi_normal.png",
    "profile_background_image_url_https": "https://.../bg.gif",
    "location": "Edmond, OK",
    "profile_use_background_image": true,
    "profile_text_color": "333333",
    "follow_request_sent": false,
...
}
```

Pretty JSON

This tool ships with Ruby 1.9 or later

```
$ curl http://twitter.com/statuses/user_timeline/JEG2.json | prettify_json.rb  
[  
 {  
   "id_str": "253982951187034113",  
   "place": null,  
   "retweeted": false,  
   "in_reply_to_status_id_str": null,  
   "in_reply_to_status_id": null,  
   "in_reply_to_screen_name": null,  
   "in_reply_to_user_id_str": null,  
   "user": {  
     "id": 20941662,  
     "profile_image_url": "http://.../fkgorpzafxmsafxp6wi_normal.png",  
     "profile_background_image_url_https": "https://.../bg.gif",  
     "location": "Edmond, OK",  
     "profile_use_background_image": true,  
     "profile_text_color": "333333",  
     "follow_request_sent": false,  
   ...  
 }
```



Pretty JSON

This tool ships with Ruby 1.9 or later

Black Magic

```
#!/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

```
#!/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

```
#!/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

```
#!/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

```

#!/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,
end

```

```

10 - 7 = 3
-1 / -6 = 0
0 + -6 = -6
-2 + -9 = -11
-8 * 1 = -8
-9 - -2 = -7
-2 + -6 = -8
-3 + 9 = 6
-3 * 9 = -27
2 / -7 = -1
...

```

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
