

Short Ruby Newsletter

code content from Issue #9

for full newsletter visit newsletter.shortruby.com

curated by [@lucianghinda](https://twitter.com/lucianghinda)

Xavier Noria [shared](#) a short explanation about how constants works in Ruby:



Xavier Noria
@fxn · [Follow](#)



Unlike variables, constants in Ruby are not their identifiers, but entries in a map conceptually stored in class and module objects.

For example,

```
Object.const_set(:X, 1)
```

creates a constant, and

```
Object.const_get(:X) # => 1
```

retrieves it. No bare 'X' was involved, see?

Source: @fxn on [Twitter](#)

Thiago Massa shared a new example of pattern matching with hashes:

```
● ● ●

# Creating a complex hash, with nested keys
my_complex_hash = {
  users: [
    {name: "Yukihiro Matsumoto", age: 57},
    {name: "Kabosu the Shiba Inu", age: 16},
    {name: "Thiago Massa", age: 33}
  ]
}

# What if we want to find the age of Matz?
my_complex_hash => {
  users: [*_, {name: "Yukihiro Matsumoto", age: matz_age}, *_]
}
matz_age # => 57

# 🤯 WTF! What happened there?

# The _ just means we want to ignore the output and not want to assign it to a variable.

# We are using the Find pattern. (experimental feature on Ruby 3.1)
# By specifying *_ in the beginning and end we can search through the data structure...
# Similar to a Regexp! 😎 So cool!

# Coming next we'll look into variable pinning in Pattern Matching 😊
```

Source: @th1agofm on [Twitter](#)

Shino Kouda shared a short code example of `.map(&:to_s)` and Brandon Weaver explain more in depth how this works:

Brandon Weaver
@keystonelemur

Replying to @ShinoKouda

For those wondering what this does:

Ampersand (&) is `to_proc`, translating to `:to_s.to_proc` which is equivalent to the following block function:

`-> v { v.to_s }`

This works with any method that takes no arguments.

Source: [@keystonelemur on Twitter](#)



Brandon Weaver @keystonelemur · Sep 6

...

Replies to [@keystonelemur](#) and [@ShinoKouda](#)

That same ampersand can also coerce other things into procs as long as they respond to `to_proc` like Hashes:

```
h = { a: 1, b: 2 }
[:a, :b].map(&h)
# => [1, 2]
```

(Hash#to_proc is basically `-> v { self[v] }`, or rather "get this key")

1

12

12

12



Brandon Weaver @keystonelemur · Sep 6

...

If you want to get _really fancy_ the precedence of `&` is lower than infix operators. See if you can get this code to work:

```
[1, 2, 3].map(&Fn + 5)
# => [6, 7, 8]
```

Hint: Closures and singleton methods.

Source: [@keystonelemur](#) [on Twitter](#)

Thiago Massa shared an example of how to use the Pin operator (^) pattern matching:

```
● ● ●

# The pin operator (^) is used in pattern matching to evaluate a variable instead of assigning it.

# That sounds complicated 🤯 We can understand it better with an example of its usage.

# Let's suppose we have an Array with some School grades
school_grades = [10, 9, 0]

# With pattern matching, we want to see if the first grade is a 10.
school_grades => [10, *]
# It matched successfully! No exception.

# What if now we want to base our lookup from a dynamic variable instead of constant 10?
expected_first_grade = 5
school_grades => [expected_first_grade, *]
expected_first_grade # => 10

# Oh no. 😱
# We have overwritten expected_first_grade with 10 instead of looking it up.

# Let's try again now and use the pin(^) operator before our expected_first_grade variable.
expected_first_grade = 5
school_grades => [^expected_first_grade, *]
# [10, 9, 0]: 5 === 10 does not return true (NoMatchingPatternError)

# Nice! ✨ It used the content of the variable as we initially desired and didn't match!

# On my next Tweet we'll learn practical ideas on where to use pattern matching.
```

Source: @th1agofm [on Twitter](#)

Joel Drapper shared an example of another underused operator, XOR:



Joel Drapper @joeldrapper · 4d

Using the XOR operator like this is really concise, and is usually clearly explained by the error message. If not, a quick search for “ruby ^ operator” reveals what it does.

```
def foo(a: nil, b: nil)
  unless !!a ^ !!b
    raise ArgumentError,
      "You must pass either a or b but not both."
  end
...
end
end
```

Source: [@joeldrapper on Twitter](#)

Lucian Ghinda shared another example of using XOR:



Lucian Ghinda
@lucianghinda

Here is also an example of a production code that I wrote in a project a while ago in the context of dry-validation:

```
rule(:after, :status) do
  if key?
    unless has_one_or_the_other?(values, :status, :after)
      key.failure("cannot filter by after and status at the same time")
    end
  end
end

private

def has_one_or_the_other?(values, key1, key2)
  values.key?(key1) ^ values.key?(key2)
end
```

Source: [@lucianghinda on Twitter](#)

[Kirill Shevchenko](#) shared two code samples explaining how |= is creating a new array instead of adding to the existing one:



Kirill Shevchenko
@kirill_shevch

|= assignment operator in [#ruby](#) (which is actually a short form of a = a | b) creates a new one array instead of adding value to an existing one, unlike push

```
● ● ●  
arr = [:one, :two]  
arr.object_id  
# => 15040  
  
arr.push :three  
arr.object_id  
# => 15040
```

```
● ● ●  
arr = [:one, :two]  
arr.object_id  
# => 21940  
  
arr |= [:three]  
arr.object_id  
# => 27480
```

Source: @kirill_shevch on [Twitter](#)

Joël Quenneville shared code about how to generate infinite series with Enumerator:

Joël Quenneville
@joelquen

#Ruby's `Enumerator.produce` is a really cool method! It can generate an infinite series where each item is used to calculate the next.

✨ For example the series of even numbers:

```
evens = Enumerator.produce(2) { |n| n + 2 }
evens.take(5)
# [2, 4, 6, 8, 10]
```

ALT

Source: [@joelquen on Twitter](#)

As the produce cannot be used to generate series where the input from each step needs the output of the previous step, then he showed how to add unfold to the Enumerator to achieve more:

Joël Quenneville @joelquen · 4d

We can make our own unfold using Enumerator's constructor

```
class Enumerator
  def self.unfold(seed, &block)
    Enumerator.new do |yielder|
      loop do
        seed, value = block.call(seed)
        raise StopIteration if value.nil?

        yielder << value
      end
    end
  end
end
```

ALT

And here is how he showed how to use that:

```
● ● ●  
squares = Enumerator.unfold(2) { |seed| [seed + 1, seed * seed] }  
squares.take(5)  
=> [4, 9, 16, 25, 36,]
```

Source: @joelquen [on Twitter](#)

Emmanuel Hayford asked a smart question: How to explain Class.class being Class:

Emmanuel Hayford
@siaw23 · [Follow](#)

How would you, *in one sentence*, explain this to a neophyte Rubyist in a way they'll understand without questions?

3.0.0 :001 > Class.class
=> Class
3.0.0 :002 >

Source: @siaw23 on [Twitter](#)

Here are some of the [examples shared](#) in that thread by [Brandon Weaver](#), [Lucian Ghinda](#), [John Cheek](#), [Ariel Caplan](#), [Eric Halverson](#):

All Ruby classes are objects of the type Class, including amusingly Class itself
(source: [@keystonelemur ↗](#))

The constant "Class" is also an object (instance of Class) yet somehow is also a class
(source: [@lucianghinda ↗](#))

You can instantiate a class with `.new`. Methods are looked up on an object's class, so `Array`. `new` finds the `new` method in `Array.class # => Class`. Further, they have set `Class.class # => Class`, so that you can programmatically create classes with `Class.new`"
(source: [@josh_cheek ↗](#))

The `Class` class is, like any class, an object whose methods are defined on that object's class, the `Class` class.
(source: [@amcaplan ↗](#))

Everything in Ruby is an object and `Class` is the mother of all objects including itself. 😊
(source: [@elhalvers ↗](#))

Brandon Weaver shared an experimental piece of code to implement proc_line:

Brandon Weaver
@keystonelemur

I should probably feel bad about this, but I don't.

ALT gist: gist.github.com/baweafer/8c826...

```
IDENTITY = -> v { v }

def proc_line(**fns)
  fns.reduce(IDENTITY) { |fn, (m, args)|
    fn >> m.to_proc.then { |m_proc| -> v { m_proc.call(v, *args) } }
  }
end

[50, 100, 300, 5000].map(&proc_line(
  to_s: 2, chars: nil, reverse: nil, join: nil, to_i: 2
))
# => [19, 19, 105, 569]
```

Source: @keystonelemur on [Twitter](#)

Brandon Weaver shared an experimental piece of code to implement proc_line:

Brandon Weaver
@keystonelemur

I should probably feel bad about this, but I don't.

ALT gist: gist.github.com/baweafer/8c826...

```
IDENTITY = -> v { v }

def proc_line(**fns)
  fns.reduce(IDENTITY) { |fn, (m, args)|
    fn >> m.to_proc.then { |m_proc| -> v { m_proc.call(v, *args) } }
  }
end

[50, 100, 300, 5000].map(&proc_line(
  to_s: 2, chars: nil, reverse: nil, join: nil, to_i: 2
))
# => [19, 19, 105, 569]
```

Source: @keystonelemur on [Twitter](#)

Jean Boussier shared their process to improve performance on string interpolation.
Amazing read!

Jean Boussier (@_byroot)

First step a micro benchmark to have a right idea of the situation. Interpolation somehow takes about 1.75x longer.

| | # Iteration per second (i/s) | compare-ruby | built-ruby |
|--------------------|------------------------------|--------------|------------|
| binary_concat_7bit | 659.391k | 649.534k | 1.02x |
| utf8_concat_7bit | 652.451k | 644.085k | 1.01x |
| utf8_concat_UTF8 | 647.227k | 647.563k | - |
| interpolation | 369.019k | 371.855k | 1.00x |
| | - | - | 1.01x |

Source: @_byroot on [Twitter](#)

[Thiago Massa](#) asked a question and shared a code sample about ampersand Ruby idiom:



Thiago Massa
@th1agofm

What's your favorite Ruby idiom?

I'll start with mine. Using &(ampersand) to write functional code in Ruby.

Methods/function in Ruby aren't first class citizens. But & make them at least second class 🎉. Code becomes shorter & looks great.

I'm curious to know yours 👇

Source: @th1agofm on [Twitter](#)

And along with this question Thiago shared a code sample:

```
● ● ●

# From:
[1,2,3].map { |n| n.odd? }
# => [true, false, true]

# To:
[1,2,3].map(&:odd?)
# => [true, false, true]

# It isn't exclusive for ruby methods
# You can also use on your own methods...

def my_method(_)
  10
end

[1,2,3].map(&method(:my_method))
# => [10, 10, 10]

# How it works?
# & calls #to_proc on the method converting it to a Proc
# Procs work pretty much like anonymous functions in Ruby
```

[Joel Drapper](#) shared a new update for [Phlex.fun](#) showing support for conditional CSS classes:



Joel Drapper
@joeldrapp

Phlex.fun now has conditional CSS classes.

```
class Link < Phlex::Component
  def initialize(text, to:, active:)
    @text = text
    @to = to
    @active = active
  end

  def template
    a @text, href: @to, **classes("nav-item",
      active?: "nav-item-active")
  end

  private

  def active? = @active
end
```

Source: [@joeldrapp](#) on [Twitter](#)

Cj Avilla shared code showing how to use right assignment / pattern matching:

The image shows a screenshot of a Twitter post from user @cjav_dev. The post includes a profile picture of a man with a beard, the name "CJ Avilla", and the handle "@cjav_dev". The main content of the post is a block of Ruby code. The code is a monkey patch for the Stripe module, specifically for the StripeObject class. It adds a deconstruct_keys method that takes a hash of keys and extracts specific values. Below this, there's an example of creating a PaymentIntent and printing its client secret. The code is as follows:

```
# Monkey patch
module Stripe
  class StripeObject
    def deconstruct_keys(keys)
      @values
    end
  end
end

Stripe::PaymentIntent.create(
  amount: 100,
  currency: 'usd'
) => { client_secret: }

p client_secret #=> "pi_3LfRKsCZ6_secret_RPlff9ie0"
```

At the bottom left of the screenshot, there is a small "ALT" button.

Source: @cjav_dev on [Twitter](#)

Matt Swanson shared code about how to use modules to organize features:



matt swanson 🐸
 @_swanson

Really digging this pattern for organizing different 'features' of a model.

Keeps related methods together and makes it easier to find relevant code as the codebase grows. Plus I can separate out test files by feature.

```
#  
class Task < ApplicationRecord  
  include Tasks::Actions  
  include Tasks::DueDate  
  include Tasks::Assignment  
  include Tasks::Visibility
```

Here are some interesting replies, but I really recommend you to [read the entire conversation](#) (for people not using Twitter use [this](#)) as it has great suggestions and some great explanation for various styles:

```
# https://twitter.com/rockatanescu proposed:  
class Task < ActiveRecord #in app/models/task.rb  
  include Actions  
end  
  
module Task::Actions # in app/models/task/actions.rb  
end  
  
# https://twitter.com/gjtorikian proposed:  
class Message < ActiveRecord  
  include Relationships::Mesage  
end  
  
module Rerlationships::Message # in app/models/relationships/message.rb  
end  
  
# https://twitter.com/ahmedabdel3aaal proposed  
class BusinessProfile < ActiveRecord  
  include BusinessProfile::Scopes  
end  
  
module BusinessProfile::Scopes # in app/models/business_profile/scope.rb  
end
```



Xavier Noria @fxn · Sep 10

...

Classes are namespaces. The important point is that object types have to agree:

```
class Post
  module DueDate
  end
end
```

As Kasper said

```
module Post::DueDate
end
```

works too.

1

1

6

1



Xavier Noria @fxn · Sep 10

...

This is not really related to the autoloaders, it is just the way Ruby works.

If you define first `class Foo`, and later say `module Foo`, Ruby is going to raise because the `Foo` constant is already defined and does not match the `module` keyword.

1

1

5

1

Source: [@fxn on Twitter](#)



Kasper Timm Hansen
@kaspth

...

Replying to [@strzibnyj](#) [@NateMatykiewicz](#) and 3 others

For me, Task::Assignment is about extending the Task from within itself, and Tasks::Assignment is about an outside module reaching into Task and extending it with something else.

For me, they're two different things because they're not about containing both controllers & models.

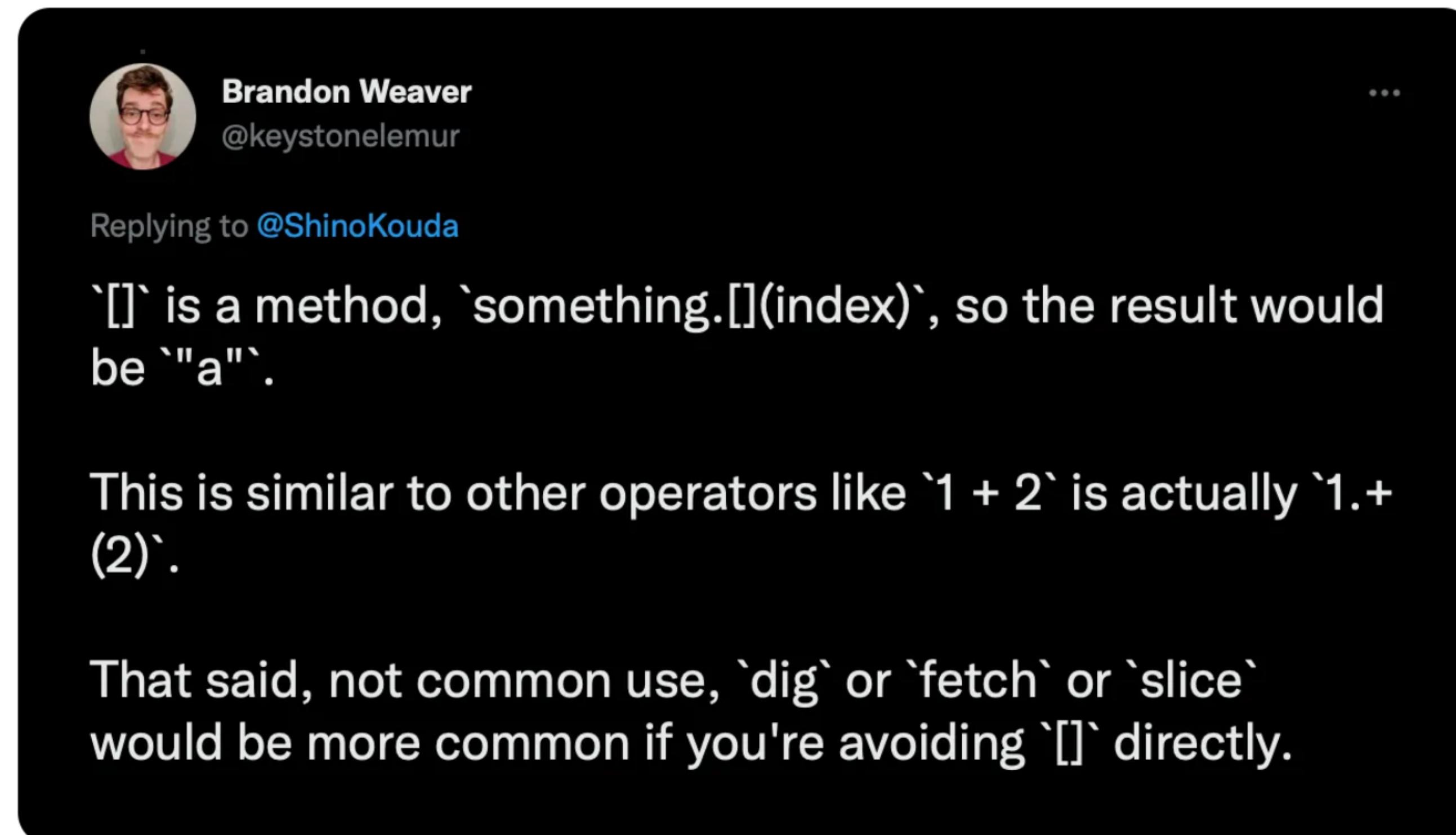
Source: [@fxn on Twitter](#)

Shino Kouda asked what is the result of executing:



```
1 [ "a", "b", "c", "d" ].[](0)
```

Brandon Weaver shared explaining how [] works for arrays and hashes:



 **Brandon Weaver**
@keystonelemur

Replying to [@ShinoKouda](#)

`'[]'` is a method, `'something.[](index)'`, so the result would be `'"a"`.

This is similar to other operators like `'1 + 2'` is actually `'1.+ (2)'`.

That said, not common use, `'dig'` or `'fetch'` or `'slice'` would be more common if you're avoiding `'[]'` directly.

[Kirill Shevchenko](#) shared code sample about how to use lazy enumerators:

Kirill Shevchenko
@kirill_shevch · [Follow](#)

Ruby's lazy enumerators are great. For e.g. you can process the data without iterating through all elements since the loop exits when N elements match the condition.

#ruby

```
# some bulky API response
# data = [...]

data.lazy
  .select { |post| post[:draft] }
  .first(3)
```

Source: @kirill_shevch on [Twitter](#)

Collin shared a tip about how to require a library when starting IRB:

```
...H=/Users/collinjilbert/.asdf/installs/ruby/3.1.2/bin:/usr/local/opt/sqlite/bin:/Users/collinjilbert/.asdf/shims:/usr/local/opt/asdf/libexec/bin:/usr/local/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin TERM_PROGRAM=Apple_Terminal TERM=xterm-256color  
[collinjilbert@Collins-MBP ~ % irb  
[irb(main):001:0> require "socket" ←  
  => true  
[irb(main):002:0> server = TCPServer.new "localhost", 3000  
  => #<TCPServer:fd 11, AF_INET6, ::1, 3000>  
[irb(main):003:0> exit  
[collinjilbert@Collins-MBP ~ %  
[collinjilbert@Collins-MBP ~ %  
[collinjilbert@Collins-MBP ~ % irb -r socket ←  
[irb(main):001:0> server = TCPServer.new "localhost", 3000  
  => #<TCPServer:fd 11, AF_INET6, ::1, 3000>  
irb(main):002:0> ]
```

Instead of writing out the full require statement inside IRB

We can instead save a bit of typing if we know ahead of time what we need by using the -r option with the name of the library

[Joe Masilitto](#) shared a sample code showing how to add hidden field to the button_to helper:

The image shows a tweet from Joe Masilitto (@joemasilitto) on a dark-themed Twitter interface. The tweet contains the following text:

You know how Rails' button_to generates a form?
TIL you can add hidden fields directly to html_options "params".
And you can set the button's CSS class!
[copy-paste code in image description]

Below the text is a code block showing the implementation:

```
<%= button_to("Create sticker", sticker_path, {  
  class: "hover:underline",  
  params: {"title" => "RailsDevs"}})  
%>  
  
<form class="button_to" method="post" action="/stickers">  
  <button class="hover:underline" type="submit">Create sticker</button>  
  <input type="hidden" name="authenticity_token" value="..." autocomplete="off">  
  <input type="hidden" name="title" value="RailsDevs" autocomplete="off">  
</form>
```

The code demonstrates how to use the `button_to` helper in Rails to generate a form with a button and two hidden input fields. The first hidden field is for the authenticity token, and the second is for the title, both with the value "RailsDevs". The button has a CSS class of "hover:underline".

At the bottom of the tweet, there is a small "ALT" button.

Source: [@joemasilitto](#) on Twitter

If you want to read the entire newsletter please go to

<https://newsletter.shortruby.com>