Ruby master - Bug #14015

Enumerable & Hash yielding arity

10/14/2017 08:19 PM - marcandre (Marc-Andre Lafortune)

Status:	Closed		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:			
ruby -v:	2.5.0 preview 1	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN

Description

The subtle difference between yield 1, 2 and yield [1, 2] has always confused me.

Today I wanted to pass a method to Hash#flat_map and realized how it's even more confusing than I thought.

I assumed that Hash#each was calling yield key, value. But somehow it's not that simple:

```
{a: 1}.map(&->(key, value){}) # => [nil]
{a: 1}.flat_map(&->(key, value){})
#=> ArgumentError: wrong number of arguments (given 1, expected 2)
```

What blows my mind, is that a custom method each that does yield a, 1 has different result!

```
class << o = Object.new
  include Enumerable
  def each
    yield :a, 1
  end
end
o.map(&->(key, value){})  # => [nil]
o.flat_map(&->(key, value){})  # => [nil]  does not raise!!
```

I don't even know how that's possible, since Hash doesn't have a specialized flat_map method...

Here's a list of methods that accept a lambda of arity 2 (as I would expect)
For Hash
each, any?, map, select, reject,
For a custom yield
each, any?, map, count, find_index, flat_map, all?, one?, none?, take_while, uniq

These two lists have each, map and any? in common. Others work in one flavor, not the other. Many require arity 1: find, sort_by, grep, grep_v, count, detect, find_index, find_all, ...

To make things even more impossible, Hash#map has been working with arity 2 since Ruby 2.4 only.

Finally, Hash#each changes the expected arity of select, reject, and any?, but not of map:

```
    {a: 1}          .select(&->(a, b){})  # => {}
    {a: 1}.each.select(&->(a, b){}) # => wrong number of arguments (given 1, expected 2)
```

Conclusion:

It seems more or less impossible to guess the expected arity of methods of Enumerable and of Hash, and they are not even consistent with one another. This makes these methods more or less unusable with lambdas.

While compatibility could be an issue, the fact that Hash#map has changed it's arity (I believe following https://bugs.ruby-lang.org/issues/13391) makes me think that compatibility with the lesser used methods would be even less of a problem.

My personal wish: that the following methods be fixed to expect arity 2 for lambdas:

For both Hash & Enumerable:

- find, sort_by, grep, grep_v, detect, find_all, partition, group_by, min_by, max_by, minmax_by, reverse_each, drop_while, sum For Hash:
- count, find_index, flat_map, all?, one?, none?, take_while, uniq For Enumerable:
- select, reject

Matz, what do you think?

History

#1 - 10/14/2017 08:26 PM - marcandre (Marc-Andre Lafortune)

- File yield_arity.rb added

I'm attaching a script I used to test this, in case it can be helpful

#2 - 10/15/2017 01:21 AM - nobu (Nobuyoshi Nakada)

- Description updated

We know, but can't fix for backward compatibilities, now.

#3 - 10/15/2017 04:40 AM - marcandre (Marc-Andre Lafortune)

Reading more, I see that in some cases there were differences between a lambda with arity 2 and a method with arity 2.
In particular, the change of Hash#map in Ruby 2.4.2 was only for methods, but lambdas were working before. What a mess.
Any hope to have all of that resolved at some point?

#4 - 10/15/2017 01:31 PM - shevegen (Robert A. Heiler)

Perhaps ruby 3.x?

To be honest, while I agree with Marc here, if only for consistency, I
think it is probably not the biggest issue overall. I use yield a lot
but most of my use cases are very simple with yield. Not that I am saying
to be representative of any real use case, either, though. :)

I guess it may be harder and more confusing for people who use lambdas
a lot.

#5 - 10/18/2017 09:16 PM - marcandre (Marc-Andre Lafortune)

Would it be possible to issue a warning for cases of lambdas with arity 1 used in 2.5, so we can go forward in the next version?

#6 - 12/04/2017 05:45 PM - marcandre (Marc-Andre Lafortune)

Matz, this was on the agenda for the developers meeting, was it discussed?

#7 - 12/12/2017 06:22 AM - matz (Yukihiro Matsumoto)

Hi,

I admit inconsistency exists. The behavior has changed time to time.
I will make the compromise to minimize the confusion. But we need some time.

Matz.

#8 - 06/01/2018 03:10 AM - marcandre (Marc-Andre Lafortune)

Here's more code to show how the situation is complicated.
I don't know which of these can be considered bugs and which are as per spec.
The method check below checks if a method accepting one argument (one) or two arguments (two) is acceptable for a particular call.

```
def check(receiver, method, expects)
  results = expects.keys.map do |block|
    receiver.send(method, &block)
    :ok
  rescue ArgumentError
    :raise
  end
  raise "Expected #{expects.values}, got #{results}" unless expects.values == results
end

hash = {a: 1}
```

```ruby
def method_one(x)
end
one = method(:method_one)

def method_two(x, y)
end
two = method(:method_two)

class << enum = Object.new
  include Enumerable
  def each
    return to_enum unless block_given?
    yield :a, 1
  end
end

### 1) Lambda vs Method

# Don't always behave the same way:

check(enum, :select,
  one           => :ok,
  (->(x) {})    => :ok,
  two           => :raise,
  (->(x, y) {}) => :ok,
) {|x| enum.select(&x) }

# (Since 2.2, https://bugs.ruby-lang.org/issues/9605)

# ### 2) Hash vs Enumerable

# Sometimes requires arity 1:
check(hash, :detect,
  one           => :ok,
  two           => :raise,
)
check(enum, :detect,
  one           => :ok,
  two           => :raise,
)


# Sometimes relaxed for Hash, but not Enumerable
check(hash, :any?,
  one           => :ok,
  two           => :ok,
)
check(enum, :any?,
  one           => :raise,
  two           => :ok,
)
# (But Hash#each no longer relaxed, now reversed from Enumerable)
check(hash.each, :any?,
  one           => :ok,
  two           => :raise,
)


# Sometimes requires arity 2 for Hash vs 1 for Enumerable
check(hash, :select,
  one           => :raise,
  two           => :ok,
)
check(enum, :select,
  one           => :ok,
  two           => :raise,
)
# (But Hash#each reverses:)
check(hash.each, :select,
  one           => :ok,
  two           => :raise,
)


# Sometimes requires the reverse: arity 1 for Hash vs 2 for Enumerable
```

```
check(hash, :find_index,
  one           => :ok,
  two           => :raise,
)
check(enum, :find_index,
  one           => :raise,
  two           => :ok,
)

# But Hash#each doesn't really behave as Enumerable...
check(hash.each, :find_index,
  one           => :ok,
  two           => :raise,
)
```

Ideally, we would have a decision as to what the correct behavior is, before we introduce a shorthand to get a method.

#9 - 09/10/2020 03:51 PM - boblail (Bob Lail)

Looks like this was resolved for map specifically in #13391 (https://github.com/ruby/ruby/commit/1f67a3900fbd45482ed36ad3b148b321307c1576)

Here are other steps to reveal the inconsistency:

```
hash = { a: "my", b: "en" }

hash.map { |k,v| [k,v].join }               # => ["amy", "ben"]
hash.map &proc { |k,v| [k,v].join }         # => ["amy", "ben"]
hash.map &lambda { |k,v| [k,v].join }       # => ["amy", "ben"]

hash.flat_map { |k,v| [k,v].join }          # => ["amy", "ben"]
hash.flat_map &proc { |k,v| [k,v].join }    # => ["amy", "ben"]
hash.flat_map &lambda { |k,v| [k,v].join }
# => ArgumentError (wrong number of arguments (given 1, expected 2))
```

#10 - 09/28/2020 02:28 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Closed

Now that Hash#each has arity 1, I'll close this and open an update in #17197

Files

yield_arity.rb	805 Bytes	10/14/2017	marcandre (Marc-Andre Lafortune)