Allow methods with `yield` to be called without a block

02/07/2016 08:23 PM - alexeymuranov (Alexey Muranov)

Status: Open
Priority: Normal
Assignee:
Target version:

Description

Trying to figure out how yield works in Python, I had the following idea.

Allow a method with yield to be called without a block. When a method encounters yield, if no block is given, the method returns an Enumerator object.

```
def f
    yield 'a'
    yield 'b'
end
e = f
puts e.next # => a
puts e.next # => b
```

It seems that this is what yield in Python does, but in Python a function with yield cannot take a block. Why not to have both?

History

#1 - 02/07/2016 08:53 PM - alexeymuranov (Alexey Muranov)

Or maybe not an iterator but a delimited continuation?

Probably the following behavior is more natural:

```
def f
    ['a', 'b', 'c'].each do |c|
        puts yield c
    end
    return 'd'
end
c, v = f
puts v # : a
c, v = c.call("#{v}") # : (a)
puts v # : b
c, v = c.call("#{v}") # : (b)
puts v # : c
v = c.call("#{v}") # : (c)
puts v # : d
c, v = f
puts v # : a
v = c.call("#{v}") #{v} "#{v}"
    # : (a)
    # [b]
    # [c]
puts v # : d
```

So by default "yield" would be just "return-with-current-delimited-continuation", but if a block is given, all references to "yield" in the method would be rebound to run that block.

By analogy to "yield from" in Python 3.3, a method yield_from can be created to work as follows:

```
def f
    [1, 2, 3].each do |i|
        yield i
    end
    nil
end
```

08/28/2022
def g
    yield_from f
    yield 0
end

c, v = g
puts v # : 1
c, v = c.call
puts v # : 2
c, v = c.call
puts v # : 3
c, v = c.call
puts v # : 0
v = c.call('The End')
puts v # : The End

"yield_from nil" does nothing.

P.S. It looks like what Python does is a bit more complicated and rather nonlogical (probably searching for "yield" keyword at the definition time).

#2 - 07/20/2016 02:15 AM - shyouhei (Shyouhei Urabe)
What OP wants can be done using enumerator.

require 'enumerator'

def f
    Enumerator.new do |y|
        y.yield 'a'
        y.yield 'b'
    end
end

e = f
puts e.next # => a
puts e.next # => b

Or, I might be failing to understand the request. Might it be "if block is present call it, but if absent no error" request? If so, following one line addition solves such request.

    return enum_for(__method__) unless block_given? # this

    yield 'a'
    yield 'b'
end

e = f
puts e.next # => a
puts e.next # => b

Either way, I see no need to extend the language.

#3 - 12/23/2021 11:43 PM - hsbt (Hiroshi SHIBATA)
- Project changed from 14 to Ruby master