Ruby - Feature #10498

Make `loop` yield a counter

11/12/2014 06:33 AM - cesario (Franck Verrot)

Status:	Assigned
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

Problem

Teaching Ruby, we always end up with that type of construct

```
i = 0
loop do
  i += 1
  # do something with i....
  raise StopIteration if i ...
end
```

Solution

What I propose with this patch is making loop yield the iteration count:

```
loop do |i|
  # do something with i....
  raise StopIteration if i ...
end
```

i starts at 0 and stops at FIXNUM_MAX (there's no Float::Infinity equivalent for integers).

Alternate solution

Integer#times could work if we had an <Integer's infinity> object, so we would just do <Integer's Infinity>.times { |i| ... }.

Also, this is the very first patch I submit to Ruby, I might have done something horrible, feel free to tell me :-)

History

#1 - 11/12/2014 03:31 PM - marcandre (Marc-Andre Lafortune)

I've also expected loop to yield a number and forget from time to time it doesn't.

Note you can achieve the same effect today with (0..Float::INFINITY).each. It's explicit, but quite a bit longer.

#2 - 11/12/2014 03:37 PM - chrisseaton (Chris Seaton)

But doesn't this mean #loop will only run FIXNUM_MAX times? Rather than run infinitely as it currently does? That's a pretty big semantic change. Also, why not just overflow to Bignum?

#3 - 11/12/2014 04:09 PM - marcandre (Marc-Andre Lafortune)

Chris Seaton wrote:

> But doesn't this mean #loop will only run FIXNUM_MAX times?

Indeed, the patch is not acceptable.

Let's consider the feature request that loop infinitely and yields a number (that will eventually be a Bignum). If this is accepted, writing the patch won't be a big issue.

I forgot another existing alternative: loop.with_index do |_, i| gets the same effect.

#4 - 11/13/2014 01:44 AM - duerst (Martin Dürst)

Shouldn't the version of loop that yields a number be called loop_with_index, to correspond with others such as each_with_index, map_with_index, and so forth? Maybe with a little bit of magic, that can be made to happen?

#5 - 11/13/2014 08:12 AM - nobu (Nobuyoshi Nakada)

- Description updated

#6 - 11/14/2014 10:28 PM - cesario (Franck Verrot)

- File 0001-vm_eval.c-loop-now-yields-a-incremented-counter.patch added

Marc-Andre Lafortune wrote:

> Chris Seaton wrote:
>
> > But doesn't this mean #loop will only run FIXNUM_MAX times?
>
> Indeed, the patch is not acceptable.

This indeed was a mistake, I've re-submitted a new patch.

So now the counter starts at 0 and will eventually become a Bignum when bigger than FIXNUM_MAX.

#7 - 11/14/2014 10:42 PM - phluid61 (Matthew Kerwin)

I agree with Martin, this should be Kernel#loop_with_index

#8 - 11/14/2014 11:35 PM - cesario (Franck Verrot)

Martin Dürst wrote:

> Shouldn't the version of loop that yields a number be called loop_with_index, to correspond with others such as each_with_index, map_with_index, and so forth? Maybe with a little bit of magic, that can be made to happen?

I might have overlooked them, but I can't find any reference to any other*_with_index method than each_with_index. Are they in Ruby core or in an external gem?

#9 - 11/15/2014 12:39 PM - joffreyjaffeux (Joffrey Jaffeux)

Franck Verrot wrote:

> I might have overlooked them, but I can't find any reference to any other*_with_index method than each_with_index. Are they in Ruby core or in an external gem?

each_with_index does exist, but concerning map, Martin was probably talking about map.with_index as defined in http://ruby-doc.org/core-2.1.5/Enumerator.html#method-i-with_index

Using loop.with_index {|i| puts i} will currently yield nil.

#10 - 11/16/2014 10:48 AM - funny_falcon (Yura Sokolov)

```
> loop.with_index.take(10)
 => [[nil, 0], [nil, 1], [nil, 2], [nil, 3], [nil, 4], [nil, 5], [nil, 6], [nil, 7], [nil, 8], [nil, 9]]
> loop.with_index{|_,i| p i; break}
0
 => nil
```

#11 - 11/16/2014 10:52 AM - funny_falcon (Yura Sokolov)

```
> LOOP = 2**1000
 => 10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376
> LOOP.times.take(10)
 => [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
> LOOP.times{|i| p i; break if i == 2}
0
1
2
```

```
=> nil
```

But still, if loop yields index it will not damage anyone.

#12 - 11/17/2014 07:51 AM - recursive-madman (Recursive Madman)

> But still, if loop yields index it will not damage anyone.

If existing code passes a lambda (or anything else with argument checking), it would damage them.

I think having loop yield a counter is very useful, but it should check #arity != 0 for backwards compatibility.

#13 - 11/17/2014 09:42 AM - funny_falcon (Yura Sokolov)

> I think having loop yield a counter is very useful, but it should check #arity != 0 for backwards compatibility.

You are right. +1

#14 - 11/17/2014 01:12 PM - znz (Kazuhiro NISHIYAMA)

How about Numeric#step?

```
>> 0.step.take(3)
=> [0, 1, 2]
>> 1.step.take(3)
=> [1, 2, 3]
```

#15 - 11/17/2014 01:38 PM - trans (Thomas Sawyer)

I always thought it would be most convenient if all loops had an intrinsic counter $i.

#16 - 11/17/2014 08:44 PM - rklemme (Robert Klemme)

I am actually against this feature. Reason: an infinite loop does not need a counter. We incur the cost of counting (especially when the figure leaves Fixnum space) on *all* infinite loops. For loops that have a fixed condition on the number (some have been shown with step take or other) that condition can be used upfront with a range or #step.

One can create a counting infinite loop like this:

```
x = Enumerator::Generator.new {|y| i = 0; loop {y << i; i += 1}}
```

or even

```
x = Enumerator::Generator.new {|y| i = -1; loop {y << (i += 1)}}
```

If needed that can be made a constant somewhere, e.g. in Enumerator.

#17 - 11/18/2014 02:31 PM - cesario (Franck Verrot)

Hi Robert, thanks for taking time reading the ticket.

Robert Klemme wrote:

> I am actually against this feature. Reason: an infinite loop does not need a counter.

Most examples I can find about it (and use personally when teaching) make use of a counter (all languages). Keeping track of the current iteration can be useful.

> We incur the cost of counting (especially when the figure leaves Fixnum space) on all infinite loops.

I definitely agree . I wasn't able to find a way to fine-tune this based on the block's arity as mentioned previously.
On the other hand, you'd need 4,611,686,018,427,387,903 iterations before paying the price of using BigNums, which compared to the code you're having in the block would probably be more expensive than incrementing a BigNum.
So I'm not sure whether or not that price should be considered high. Anything I'm missing?

```
x = Enumerator::Generator.new {|y| i = -1; loop {y << (i += 1)}}
```

> If needed that can be made a constant somewhere, e.g. in Enumerator.

Given the context, I'm afraid I wouldn't use this unless it's a well-known constant.

Thanks again, looking forward to more insights :-)

#18 - 11/18/2014 02:47 PM - rklemme (Robert Klemme)

Franck Verrot wrote:

> Hi Robert, thanks for taking time reading the ticket.

You're welcome!

> Robert Klemme wrote:
>
> > I am actually against this feature. Reason: an infinite loop does not need a counter.
>
> Most examples I can find about it (and use personally when teaching) make use of a counter (all languages). Keeping track of the current iteration can be useful.

"can"!

> > We incur the cost of counting (especially when the figure leaves Fixnum space) on all infinite loops.
>
> I definitely agree . I wasn't able to find a way to fine-tune this based on the block's arity as mentioned previously.

That seems fairly easy:

```
def lp(&b)
  return to_enum(:lp) unless b

  if b.arity == 0
    while true
      b[]
    end
  else
    i = 0

    while true
      b[i]
      i += 1
    end
  end

  raise "This must never happen"
end
```

> On the other hand, you'd need 4,611,686,018,427,387,903 iterations before paying the price of using BigNums, which compared to the code you're having in the block would probably be more expensive than incrementing a BigNum.

Yes, but as a user of loop you would not have a choice any more. I'd rather use the approach from above or define another method loop_with_index or use the approach with Generator and a constant. I would definitively not unconditionally provide a counter.

> So I'm not sure whether or not that price should be considered high. Anything I'm missing?

I think it is generally bad to do unnecessary work. Also it is inelegant. :-)

#19 - 11/18/2014 04:24 PM - cesario (Franck Verrot)

Robert Klemme wrote:

> > [...] I wasn't able to find a way to fine-tune this based on the block's arity as mentioned previously.
>
> That seems fairly easy:

```
def lp(&b)
  return to_enum(:lp) unless b

  if b.arity == 0
    while true
      b[]
    end
  else
    i = 0

    while true
      b[i]
      i += 1
    end
  end

  raise "This must never happen"
end
```

I meant in C but yes, given we could have Kernel#loop_with_index implemented in Ruby, it would be a no-brainer.

> On the other hand, you'd need 4,611,686,018,427,387,903 iterations before paying the price of using BigNums, which compared to the code you're having in the block would probably be more expensive than incrementing a BigNum.

> Yes, but as a user of loop you would not have a choice any more. I'd rather use the approach from above or define another method loop_with_index or use the approach with Generator and a constant. I would definitively not unconditionally provide a counter.

I get your point.

times does yield a counter, would you say it's the same concern?

#20 - 11/18/2014 04:33 PM - rklemme (Robert Klemme)

Franck Verrot wrote:

> Robert Klemme wrote:
>
> > Yes, but as a user of loop you would not have a choice any more. I'd rather use the approach from above or define another method loop_with_index or use the approach with Generator and a constant. I would definitively not unconditionally provide a counter.
>
> I get your point.
>
> times does yield a counter, would you say it's the same concern?

No, because that is specifically designed for iterating a fixed number of times.

#21 - 12/31/2014 06:29 AM - duerst (Martin Dürst)

- Assignee changed from core to matz (Yukihiro Matsumoto)

#22 - 01/05/2018 09:01 PM - naruse (Yui NARUSE)

- Target version deleted (2.2.0)

#23 - 04/03/2024 03:50 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned

Files

0001-vm_eval.c-loop-now-yields-a-incremented-counter.patch	1.74 KB	11/12/2014	cesario (Franck Verrot)
0001-vm_eval.c-loop-now-yields-a-incremented-counter.patch	1.86 KB	11/14/2014	cesario (Franck Verrot)