Ruby master - Feature #12912

An endless range `(1..)`

11/09/2016 02:57 PM - mame (Yusuke Endoh)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	2.6

Description

Why don't you allow a range without end, like (1..)?

There are two advantages. First, we can write ary[1..] instead of ary[1..-1]. The -1 is one of the most I dislike in Ruby. It is very magical, ugly, redundant, and disappointing. I envy Python's ary[1:]. I know that ary.drop(1) is slightly fast, but too long for such a common operation, IMO.
Second, we can write (1..).each {|n| ... }. As far as I know, there is no simple way in Ruby to write an endless loop with index. Which do you like among these?

```
(1..).each {|n| ... }
n = 1; loop { ...; n += 1}
1.upto(Float::INFINITY) {|n| ... }
```

Contrary to my expectation, this syntax extension causes no parser conflict. A patch is attached (for proof of concept). It requires more work, for example, (1..).step(2){} is not supported yet. But I'd like to hear your opinion first.

What do you think?

Side remarks

I don't like ary[1..-2] so much, but it is actually needed because ary[1..ary.size-2] is absurdly long.
Some people may prefer ary[1..-1] because of consistency to ary[1..-2]. However, I don't think it is reasonable to force a user who just want to take a suffix of an array, to use the dirty hack of negative index.
I don't think ary[1...] (exclusive) is meaningful.
It is better to have ary[..1] as a consistency. But it will cause a shift/reduce conflict. Anyway, ary[0..1] looks not so bad to me since it have no cursed negative index. So I don't push it.

How to implement

In the case if this proposal is accepted, what I concern is its semantics: what (1..) should return? I have three candidates.

1) Equivalent to (1..-1). This is the simplest: ary[1..] will work great with no change. We just have to fix Range#each (and other relevant methods).
However, it may be inconvenient to distinguish (1..) from (1..-1). For example, (1..-1).each {} will also loop, which is an incompatibility.
Also, we must decide how we handle ("foo"..).

2) Equivalent to (1..nil). We must modify both Array#[] and Range#each, but the change will rarely cause incompatibility.
An existing method that does not know this proposal may not work correctly (such as raising an exception), but won't cause any catastrophic failure (such as core dump).

3) Introduce a new type of Range object, such as (1..Qundef). This is the most radical way. An old method that does not know the type of Range will cause abnormal termination.

IMO, 2) is the most reasonable. The attached patch uses 1) as a proof of concept.

Note that (1..nil) is not a brand new type of Range. It is traditionally possible to create (1..nil) in pure Ruby, you know:

```
p Marshal.load("\x04\bo:\nRange\b:\texclF:\nbegini\x06:\bend0")
#=> (1..nil)
```

A piece of cake :-) So, to be precise, an incompatibility issue may occur if any method uses (1..nil) meaningfully. I believe that there is no such a method, though.

Related issues:	
Related to Ruby master - Bug #12915: 1..nil can be created by Marshal.load	**Closed**
Related to Ruby master - Feature #14697: Introducing Range#% as an alias to R...	**Closed**
Related to Ruby master - Bug #14699: Subtle behaviors with endless range	**Closed**

Associated revisions

Revision ad5a6aa7 - 04/20/2018 12:10 AM - nobu (Nobuyoshi Nakada)

range.c: fix fixnum loop condition

- range.c (range_step): FIXABLE + FIXABLE never overflow, but may not be FIXABLE. [Feature #12912]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63207 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63207 - 04/20/2018 12:10 AM - nobu (Nobuyoshi Nakada)

range.c: fix fixnum loop condition

- range.c (range_step): FIXABLE + FIXABLE never overflow, but may not be FIXABLE. [Feature #12912]

Revision 63207 - 04/20/2018 12:10 AM - nobu (Nobuyoshi Nakada)

range.c: fix fixnum loop condition

- range.c (range_step): FIXABLE + FIXABLE never overflow, but may not be FIXABLE. [Feature #12912]

Revision db885d08 - 04/20/2018 12:23 AM - nobu (Nobuyoshi Nakada)

range.c: step in bignum

- range.c (range_step): honor step in bignum addition. [Feature #12912]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@63208 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 63208 - 04/20/2018 12:23 AM - nobu (Nobuyoshi Nakada)

range.c: step in bignum

- range.c (range_step): honor step in bignum addition. [Feature #12912]

Revision 63208 - 04/20/2018 12:23 AM - nobu (Nobuyoshi Nakada)

range.c: step in bignum

- range.c (range_step): honor step in bignum addition. [Feature #12912]

History

#1 - 11/09/2016 07:32 PM - knu (Akinori MUSHA)

Yusuke Endoh wrote:

> Second, we can write (1..).each {|n| ... }. As far as I know, there is no simple way in Ruby to write an endless loop with index. Which do you like among these?
>
> ```
> (1..).each {|n| ... }
> n = 1; loop { ...; n += 1}
> 1.upto(Float::INFINITY) {|n| ... }
> ```

You could say 1.step {|n| ... }.

#2 - 11/10/2016 03:27 AM - mame (Yusuke Endoh)

- Related to Bug #12915: 1..nil can be created by Marshal.load added

#3 - 11/10/2016 03:33 AM - mame (Yusuke Endoh)

Akinori MUSHA wrote:

> You could say 1.step {|n| ... }.

Thank you, I completely forgot it. I think step is indeed good to have, but the name does not feel the intention of infinite loop to me. In this regard,

(1..).each represents my intention well, in my personal opinion.

#4 - 11/13/2016 09:02 AM - shyouhei (Shyouhei Urabe)

JFYI I found a log of IRC conversation about this topic in Japanese, done 6 years ago: http://naruse.hateblo.jp/entry/20100907/1283848281
I think Yusuke was in the IRC channel back then.

#5 - 01/19/2017 07:18 AM - akr (Akira Tanaka)

- Status changed from Open to Feedback

I think Integer#step and Array#drop is enough for many situations.

I agree these methods are not so intuitive.
But I think this non-intuitiveness is not enough for syntax extension.

#6 - 04/18/2018 04:47 PM - shevegen (Robert A. Heiler)

The idea is interesting (I think we currently can not specify
infinite ranges easily? Then again I myself have not yet had a
need to do so either); I am not sure if the syntax is good,
though.

#7 - 04/19/2018 06:19 AM - matz (Yukihiro Matsumoto)

The syntax appears a bit weird but far better than 1..Float::Infinity.
I accept it.

Matz.

#8 - 04/19/2018 06:49 AM - matz (Yukihiro Matsumoto)

- Related to Feature #14697: Introducing Range#% as an alias to Range#step added

#9 - 04/19/2018 03:29 PM - mame (Yusuke Endoh)

- Target version set to 2.6

- Status changed from Feedback to Closed

Thank you matz. I've committed this at r63192..r63197. The implementation uses Approach 2, i.e., now we can create (1..nil), a range whose end is nil.

I made Range's methods reasonably support endless range, at least, as far as I have noticed. But it is arguable about some methods. I'll create another ticket.

#10 - 04/19/2018 03:41 PM - mame (Yusuke Endoh)

- Related to Bug #14699: Subtle behaviors with endless range added

#11 - 04/27/2018 06:24 AM - sowieso (So Wieso)

I like this change.

As for general ranges:
I was somewhat comfortable with .step, but this is much nicer. Having a short global variable for Float::INFINITY would be fine too, imho. Something like 0..Inf or 0..INF.

As for array slicing:
This is the part that is even more interesting, as it's something that gets used all of the time.

> It is better to have ary[..1] as a consistency. But it will cause a shift/reduce conflict. Anyway, ary[0..1] looks not so bad to me since it have no cursed negative index. So I don't push it.

I don't see the conflict. Can you explain what you mean? I actually mind the inconsistency more than the negative index (never had a problem with it).

Not having the opposite (..5 and ..-2) feels like this is rather a hack than a thoroughly planned feature. I think everyone would try it out and be confused that it's not working in both directions. Are you sure it's not possible to do it both ways? Just a short hand like this: ..x => nil..x => 0..x (the nil step might not even be needed).

For completeness, though I think it might not be needed, what about (..)? (..).each won't iterate (like [].each), a[..] would copy the array (like (0..-1)).

#12 - 04/27/2018 07:19 AM - duerst (Martin Dürst)

sowieso (So Wieso) wrote:

> It is better to have ary[..1] as a consistency. But it will cause a shift/reduce conflict. Anyway, ary[0..1] looks not so bad to me since it have no cursed negative index. So I don't push it.

I don't see the conflict. Can you explain what you mean?

A shift/reduce conflict is something that occurs in the parser.

I actually mind the inconsistency more than the negative index (never had a problem with it).

Not having the opposite (..5 and ..-2) feels like this is rather a hack than a thoroughly planned feature.

I don't understand the need for a ..5 Range. The feature is called "endless range". Although mathematically, it's possible to think about startless ranges, they don't work in a program. Maybe some programming languages have ..5 as a shortcut for 0..5, but that's in any way a usual, bounded, range with a start and an end. It's conceptually totally different from 5.., which is a range with a start but no end, an unbound range.

I think everyone would try it out and be confused that it's not working in both directions.

I can imagine that some people might try it for Array subscripts, to get an array slice. But it doesn't make sense when used as a freestanding notation.

For completeness, though I think it might not be needed, what about (..)? (..).each won't iterate (like [].each),

No, (..).each would iterate from 0 on (to infinity, at least in theory). It's different from [].each.

a[..] would copy the array (like (0..-1)).

Which means that it's pretty much useless, because a.dup would be clearer.

#13 - 04/27/2018 07:44 AM - sowieso (So Wieso)

I tried to structure my post to make it clear that there a two different types of uses for ranges. Of course ..5 where the left bound is -∞, wouldn't make sense for iterating. But being able to skip the right bound, but not the left, breaks symmetry for array slices. That's why ..5 should be a shortcut for 0..5 imho. For consistency a ...5 (== ..4) should exist too (and 5.. == 5...).

#14 - 04/28/2018 01:25 PM - nobu (Nobuyoshi Nakada)

Why is (1..).size nil, but not Float::INFINITY?

#15 - 04/28/2018 03:55 PM - marcandre (Marc-Andre Lafortune)

nobu (Nobuyoshi Nakada) wrote:

> Why is (1..).size nil, but not Float::INFINITY?

It's a bug, see #14699

Files

endless-range.patch	943 Bytes	11/09/2016	mame (Yusuke Endoh)