Ruby master - Feature #4541
Inconsistent Array.slice()
03/31/2011 04:44 PM - kbi (Marcin Pietraszek)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>matz (Yukihiro Matsumoto)</td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

Description
Array slice/[] method is a bit inconsistent. Is it just poorly documented "feature" or a bug? In API doc I can't find this behaviour mentioned as a "special case".

```ruby
def test_array_slice
  array = ['a', 'b', 'c']
  assert_equal nil, array[3]
  assert_equal nil, array[4]

  assert_eaual [], array[3, 0] #
  assert_equal nil, array[4, 0] # [] expected (or both nils in array[3, 0] and array[4, 0])

  assert_equal ['c'], array[2..2]
  assert_equal [], array[3..3] #
  assert_equal nil, array[4..4] # [] expected (or both nils in array[3..3] and array[4..4])
end
```

Same behaviour can be reproduced on ruby 1.8.7 (2010-12-23 patchlevel 330) [x86_64-linux].

Related issues:
Is duplicate of Ruby master - Bug #4245: Array index with Range inconsistency

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

Description
Array slice/[] method is a bit inconsistent. Is it just poorly documented "feature" or a bug? In API doc I can't find this behaviour mentioned as a "special case".

```ruby
def test_array_slice
  array = ['a', 'b', 'c']
  assert_equal nil, array[3]
  assert_equal nil, array[4]

  assert_eaual [], array[3, 0] #
  assert_equal nil, array[4, 0] # [] expected (or both nils in array[3, 0] and array[4, 0])

  assert_equal ['c'], array[2..2]
  assert_equal [], array[3..3] #
  assert_equal nil, array[4..4] # [] expected (or both nils in array[3..3] and array[4..4])
end
```

Same behaviour can be reproduced on ruby 1.8.7 (2010-12-23 patchlevel 330) [x86_64-linux].

Related issues:
Is duplicate of Ruby master - Bug #4245: Array index with Range inconsistency

History
#1 - 04/05/2011 04:04 PM - naruse (Yui NARUSE)
  - Status changed from Open to Assigned
  - Assignee set to matz (Yukihiro Matsumoto)

The fix may be following but it can be a spec...

diff --git a/array.c b/array.c
index bdeb768..4721387 100644
--- a/array.c
+++ b/array.c
@@ -948,7 +948,7 @@ rb_ary_subseq(VALUE ary, long beg, long len)
 |
 | VALUE klass;
 - if (beg > RARRAY_LEN(ary)) return Qnil;
+ if (beg >= RARRAY_LEN(ary)) return Qnil;
   if (beg < 0 || len < 0) return Qnil;

   if (RARRAY_LEN(ary) < len || RARRAY_LEN(ary) < beg + len) {
```

#2 - 04/08/2011 03:01 AM - marcandre (Marc-Andre Lafortune)

Hi,

On Tue, Apr 5, 2011 at 3:04 AM, Yui NARUSE redmine@ruby-lang.org wrote:

Issue #4541 has been updated by Yui NARUSE.

Status changed from Open to Assigned
Assignee set to Yukihiro Matsumoto

The fix may be following but it can be a spec...
diff --git a/array.c b/array.c
index bdeb768..4721387 100644
--- a/array.c
+++ b/array.c
@@ -948,7 +948,7 @@

   if (beg > RARRAY_LEN(ary)) return Qnil;
 + if (beg >= RARRAY_LEN(ary)) return Qnil;
   if (beg < 0 || len < 0) return Qnil;
   if (RARRAY_LEN(ary) < len || RARRAY_LEN(ary) < beg + len) {
     VALUE klass;

With all due respect, I am strongly opposed to that change. There is no way that the following results should change:

array[0...3] # => ["a", "b", "c"]
array[1...3] # => ["b", "c"]
array[2...3] # => ["c"]
array[3...3] # => [], as it "must" be.

The only possible change would be that array[42..42] returns [] instead of nil. For consistency's sake, String#slice would also have to be changed. I am not in favor of that change either, for compatibility reason and because I feel the current API makes sense. See my comment on issue #4245 [ruby-core:34197]. This issue is a duplicate of #4245.

#3 - 04/12/2011 09:10 PM - zimbam (zimba tm)
I don't see the advantage of having nil returned in any case since the empty array already expresses the "there is no object in that range".

Out of bound can be tested separately if necessary, but most of the cases you just want to get a range and a resulting array. Having also nil being returned means that you need some more code to test return.nil? && return.empty?

#4 - 06/26/2011 04:13 PM - nahi (Hiroshi Nakamura)
- Target version changed from 1.9.2 to 1.9.3

#5 - 06/29/2011 12:54 PM - nobu (Nobuyoshi Nakada)
This is not a bug.
I'm against the spec change of this behavior.

#6 - 06/29/2011 08:28 PM - rosenfeld (Rodrigo Rosenfeld Rosas)
Nobuyoshi, could you explain why you don't find the current behavior confusing and would like to keep it as it is? What would be a useful use case for the current implementation?

#7 - 06/30/2011 04:49 AM - gwright (Gary Wright)
This discussion might help explain the rational for the current behavior: http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/380636

There is also a related rationale in this posting:

#8 - 06/30/2011 06:46 AM - marcandre (Marc-Andre Lafortune)
- Tracker changed from Bug to Feature

#9 - 06/30/2011 06:47 AM - marcandre (Marc-Andre Lafortune)
- Category set to core
- Target version changed from 1.9.3 to 2.0.0

Changing to "feature" as this is a spec change.

As I stated earlier, I'm against either propositions.

#10 - 06/30/2011 11:09 AM - rosenfeld (Rodrigo Rosenfeld Rosas)
What I think is less confusing is to always return an array when passing ranges, like the following example:

[][1..2] #=> [] instead of nil

Think in some practical method like this one:
Yes, I know this is not a great example, but I'm too tired right now to think in a better one...

**#11 - 10/30/2011 07:24 AM - alexeymuranov (Alexey Muranov)**

I better remove my previous comment, it was not a good idea. I agree that there seem to be an inconsistency between [1,2,3][3..1] # => [] and [1,2,3][4..1] # => nil.

In my opinion, the confusion about the behavior of Array#slice could be coming from the difficulties in choosing good definitions for Range and Array. For example, the ranges (3..1), (4..1), (1..-1) all have no elements, but can be used to slice an array, all in different ways.

It could be easier if there existed only one empty range: (2..1) == (1..-1) and (2..1) == (1...1) == (z..'a') # => true. But then there would come the problem of not being able to slice an array from the beginning and the end simultaneously with a[1...-2]. This can be resolved by introducing RelativeRange and RelativeNumber and do like this: a[1.from_bottom..(-1).from_top]. (The method #include? would not be defined for RelativeRange and relative numbers with different "anchors" would not be comparable; "anchor" would be an arbitrary symbol, like :top or :bottom.) Just an idea.

To me, array is an object between string and hash, so how about viewing array as a special kind of hash, where only integer keys are allowed, and the keys come in pairs ([1,2,3][1] == [1,2,3][-2]), and think from there? (Strings should not probably be viewed as hash because splitting into characters is not very well defined for accented or otherwise decorated strings.) These are just my ideas.

-Alexey.

Update: i made a feature request #5534 for RelativeNumeric and RelativeRange.

**Last edited 2011-11-01**

**#12 - 11/20/2012 08:52 PM - mame (Yusuke Endoh)**

- Target version changed from 2.0.0 to 2.6

**#13 - 12/12/2012 01:27 AM - alexeymuranov (Alexey Muranov)**

I have proposed a solution in #7546.

**#14 - 12/12/2012 04:42 AM - stomar (Marcus Stollsteimer)**

Regarding the OP's criticism of poor API documentation, this has in the meantime been improved (with issue #6680).

From the current documentation for Array#slice:

Negative indices count backward from the end of the array (-1 is the last element). For +start+ and +range+ cases the starting index is just before an element. Additionally, an empty array is returned when the starting index for an element range is at the end of the array.

Returns +nil+ if the index (or starting index) are out of range.

```ruby
a = ['a', 'b', 'c', 'd', 'e']
a[2] + a[0] + a[1] #=> "cab"
a[6]           #=> nil
a[1, 2]       #=> ['b', 'c']
a[1..3]       #=> ['b', 'c', 'd']
a[4..7]       #=> ['e']
a[6..10]      #=> nil
a[-3, 3]      #=> ['c', 'd', 'e']
```

**special cases**

```ruby
a[5]       #=> nil
a[6, 1]    #=> nil
a[5, 1]    #=> []
a[5..10]   #=> []
```

08/28/2022
Regarding the OP’s criticism of poor API documentation, this has in the meantime been improved (with issue #6680).

I hope this does not mean that the specification is now based on the implementation :).

In my opinion,

```ruby
array = ['a', 'b', 'c']
array[4..4] # => []
```

makes better sense than

```ruby
array = ['a', 'b', 'c']
array[4..4] # => nil
```

I think that nil should only be returned if there is no way to return a meaningful value, but also no particular reason to raise an error. Here a meaningful value to return IMO is [], the same as for array[3..4].

Think of a line of small tiles with letters 'a', 'b', 'c' on them which you have enumerated in your head starting with 0 on the left.

If you are asked to form a new line with letters 1..5, you will get the line [b', 'c'] by taking 'b', then 'c', then nothing, and then taking nothing 2 more times.
If you are asked to form a new line with letters 3..4, you will get the empty line by taking 2 times nothing.
If you are asked to form a new line with letters 4..4, you will get the empty line by taking 1 time nothing.
If you are asked to form a new line with letters 4..5, you will get the empty line by taking 2 times nothing.

You will always get a line, you will not get "nil".

Actually it kind of is. Hence the term "reference implementation."

One other thing to consider is that nil is falseish, but [] is trueish.

```ruby
!!nil #=> false
!![]  #=> true
```

Your change could break existing logic. I know it's relatively trivial to use `((array[range].empty?))` but that's still a change that every developer potentially needs to make to the existing codebase.

I just wanted to point out that the docs now reflect that it is intended behavior and not a bug.
Nothing more, nothing less.
We don't want to change the behavior of basic methods. That would break existing code.
FYI, the slice is nil when the starting point does not fit in the array.

Matz.