Ruby master - Misc #16678

Array#values_at has unintuitive behavior when supplied a range starting with negative index

03/07/2020 06:21 AM - prajjwal (Prajjwal Singh)

Status:	Open	
Priority:	Normal	
Assignee:		

Description

Consider the following:

```
# frozen_string_literal: true

a = (1..5).to_a

p a.values_at(3..5) # => [4, 5, nil]
p a.values_at(-1..3)  # => []
```

When the range begins with a negative (-1, 0, 1, 2, 3), it returns an empty array, which surprised me because I was expecting [1, 2, 3, 4].

The argument for this is that it cold be confusing to allow this because the index -1 could refer to the last argument and it would be unintuitive to return an array [5, 1, 2, 3, 4] with jumbled values.

The argument against it is that it makes perfect sense to account for this case and return [nil, 1, 2, 3, 4].

Opening a dialog to see what others think of this.

History

#1 - 03/07/2020 10:42 AM - shevegen (Robert A. Heiler)

Actually .values_at() confused me when I tried to use my go-to method for obtaining a slice from an Array:

```
a[3..5] # => [4, 5]
```

There I wondered why it did not return the same. :-)

But anyway; I believe the question is what -1 refers to. It should be the last entry, right? Ok, so what should the 3 indicate? I think you reason that it should refer to the fourth entry (I think ... if an Array count begins at 0, then 3 would refer to the fourth entry). So from that point of view I actually do not even disagree with you; perhaps I may have missed some other explanation. (There is probably another explanation; I think this has come up in the past too. I forgot the explanation, though, if there was one.)

Personally I will stick with [] an leave .values_at() to others. I am just so used to [] there. ;-)

#2 - 03/07/2020 02:23 PM - Eregon (Benoit Daloze)

You can easily achieve wrap-around behavior with:

```
> (1..5).to_a.values_at(*(-1..3))
=> [5, 1, 2, 3, 4]
```

Using a Range for values_at is like taking a slice with Array#[]/slice, and Array slices never wrap around (a good thing IMHO, that would be expensive to compute and confusing).

#3 - 03/09/2020 02:06 PM - Dan0042 (Daniel DeLorme)

Negative indices have always meant "offset from the end" in ruby. So if you take a negative index and add the size of the array you get the "normal index" and then I think you'll see everything is pretty intuitive.

```
a = (1..5).to_a

# get all values from a[-4] (a[1]) to a[3]
```

```
a.values_at(-4..3) #=> [2, 3, 4]
a.values_at(1..3)  #=> [2, 3, 4]

# get all values from a[-1] (a[4]) to a[3]
a.values_at(-1..3) #=> []
a.values_at(4..3)  #=> []  #range start > range end = empty range, therefore empty array
```

But I think this is slightly inconsistent:

```
(4..6).map{ a[_1] } #=> [5, nil, nil]
a.values_at(4..6)   #=> [5, nil, nil]

(-7..-5).map{ a[_1] } #=> [nil, nil, 1]
a.values_at(-7..-5)   #=> RangeError (-7..-5 out of range), should be [nil, nil, 1] imho
```