Ruby master - Feature #16665

Add an Array#except_index method

02/29/2020 02:46 PM - alex_golubenko (Alex Golubenko)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

The main idea is to implement a method that we can use to exclude elements from the array by their indices.

For example:

```
%w( a b c d e f).except_index(0, -1)
=> ["b", "c", "d", "e"]

%w( a b c d e f g h ).except_index(0..1, 3, -2..-1)
=> ["c", "e", "f"]
```

I was meeting many questions on the StackOverflow about how to do such functionality also found many topics about it.
So I think it might a helpful addition.

I spent a few days finding the proper solution on Ruby that might be acceptable with integers and ranges(both positive and negative) and has good performance:

```
  def except_index(*indexes)
    indexes.each_with_object(dup) do |ind, obj|
      ind.is_a?(Range) ? ind.each { |i| obj[i] = false } : obj[ind] = false
    end.select(&:itself)
  end
```

As you can see it's have not the best readability so I think it's a good point to add a built-in method on C.

History

#1 - 02/29/2020 06:05 PM - shevegen (Robert A. Heiler)

I can not comment on how useful this may be; I think I may have added something similar
once in a while, but I can not say how useful or common that would be. IMO we should
see to determine how common the above idiom is, before we can say that it should be
part of core ruby. (This is not necessarily a con-opinion from me, just that I think
it may be better to assess how useful the addition would be, objectively).

There is one minor complaint that I have and I think it is about the name. The
word "index" is used a lot so I guess we can expect a method that has "index" to
do something with ... an index, or indici/indices - no problem there. But "except"
is quite rare in ruby. Python has try/except clause. I believe that "except" as
word is not completely fitting into ruby's parlance, as-is. I could be wrong :)
but it does not "feel" very ruby-ish.

I can not offer a better alternative, but if we look at other ruby-method names,
we have select, reject, filter ... and .each_with_index, and similar method
names. It may be best to be slowly considering whether "except" may be a good
addition - not necessarily saying that it may be bad, but I am not sure it is
a great name. (The name can be changed if the underlying suggestion has merit,
so perhaps we should first evaluate whether the functionality is useful,
before finding an alternative name, or sticking to the suggested name.)

#2 - 03/01/2020 07:03 AM - sawa (Tsuyoshi Sawada)

For the positive counterpart, we have Array#values_at. I think the method name to be proposed should be somehow aligned with that.

#3 - 03/02/2020 09:01 AM - p8 (Petrik de Heus)

Isn't this easily solvable with select and with_index?

```
%w( a b c d e f).select.with_index{|l,index| index > 1 } # =>  ["c", "d", "e", "f"]
```

#4 - 03/02/2020 07:18 PM - alex_golubenko (Alex Golubenko)

shevegen (Robert A. Heiler) wrote in #note-1:

> I can not comment on how useful this may be; I think I may have added something similar
> once in a while, but I can not say how useful or common that would be. IMO we should
> see to determine how common the above idiom is, before we can say that it should be
> part of core ruby. (This is not necessarily a con-opinion from me, just that I think
> it may be better to assess how useful the addition would be, objectively).
>
> There is one minor complaint that I have and I think it is about the name. The
> word "index" is used a lot so I guess we can expect a method that has "index" to
> do something with ... an index, or indici/indices - no problem there. But "except"
> is quite rare in ruby. Python has try/except clause. I believe that "except" as
> word is not completely fitting into ruby's parlance, as-is. I could be wrong :)
> but it does not "feel" very ruby-ish.
>
> I can not offer a better alternative, but if we look at other ruby-method names,
> we have select, reject, filter ... and .each_with_index, and similar method
> names. It may be best to be slowly considering whether "except" may be a good
> addition - not necessarily saying that it may be bad, but I am not sure it is
> a great name. (The name can be changed if the underlying suggestion has merit,
> so perhaps we should first evaluate whether the functionality is useful,
> before finding an alternative name, or sticking to the suggested name.)

I also thought a lot about naming and have a few variants:

`omit_indices`

`omit_index`

`values_out`

#5 - 03/02/2020 07:19 PM - alex_golubenko (Alex Golubenko)

sawa (Tsuyoshi Sawada) wrote in #note-2:

> For the positive counterpart, we have Array#values_at. I think the method name to be proposed should be somehow aligned with that.

May I ask you to check my answer to Robert A. Heiler?

#6 - 03/02/2020 09:42 PM - schwad (Nick Schwaderer)

Hello, I was part of this discussion on Railstalk and the Rails-core issue, you can peek for additional context here:
https://github.com/rails/rails/pull/38611#issuecomment-593627540

It was closed on Rails-core and suggested to be proposed here. I also tried to help with the naming because it was originally #except to match Hash#except in Rails (which has also been recommended to be extracted to Ruby-core).

I am in favor of this proposal, even if the name is tweaked, or we include Hash#except as well.

I think omit_indices is super interesting- if you saw [1, 2, 3, 4, 5].omit_indices(2, 4) in the wild I'm sure you'd have no doubt what it did.

#7 - 03/07/2020 12:58 AM - prajjwal (Prajjwal Singh)

In my opinion omit_indices is the best name in the thread so far, but I'm not a huge fan of making my methods verbs. I propose arr.except_indices(1, 2, 3) as an alternative.

The negation of values_at() is a semi-common occurrence and I'd love to have it in ruby core. I'll try to throw something together today, it can be trivially renamed whenever matz chimes in.

#8 - 03/07/2020 06:39 AM - prajjwal (Prajjwal Singh)

There needs to be more discussion on the behavior of this method.

Logically, except_indices should be the complement of values_at, so for the purpose of writing test cases I added this stopgap:

```
def except_indices(*idx)
  self - self.values_at(*idx)
end
```

And ended up with following test cases:

```
def test_except_indices
```

```
  a = @cls['a', 'b', 'c', 'd', 'e']

  assert_equal(@cls['a', 'c'], a.except_indices(1, -2, -1))
  assert_equal(@cls['c', 'd', 'e'], a.except_indices(0, 1, 10))
  assert_equal(@cls['a', 'b', 'c', 'd', 'e'], a.except_indices(10, 20, 30))

  assert_equal(@cls[], a.except_indices(0..4))
  assert_equal(@cls['c', 'd', 'e'], a.except_indices(0..1))
  assert_equal(@cls['c'], a.except_indices(0..1, 3..4))
  assert_equal(@cls['a', 'b', 'c'], a.except_indices(3..15))
  # Logically except_indices is the complement of values_at, and values_at
  # returns an empty array when given a range starting with a negative number.
  # This could change in the future.
  # https://bugs.ruby-lang.org/issues/16678
  assert_equal(@cls['a', 'b', 'c', 'd', 'e'], a.except_indices(-1..2))
end
```

However, I'm left wondering whether that last case is something that should be fixed (with values_at being changed to accomodate ranges starting with a negative), or if this is fine.

Any feedback welcome. Full commit is at - https://github.com/Prajjwal/ruby/commit/0bb2788fb249df4381982cea957c7feaadccd0ed.

#9 - 03/09/2020 01:31 PM - Dan0042 (Daniel DeLorme)

That implementation would return an empty array for [true,nil,nil].except_index(0)
Probably something a bit more like this:

```
  @@undef = Object.new #using Qundef in C
  def except_index(*indexes)
    result = dup
    indexes.each do |ind|
      if ind.is_a?(Range)
        ind.each{ |i| result[i] = @@undef }
      else
        result[ind] = @@undef
      end
    end
    result.reject!{ |e| e == @@undef }
    result
  end
```

It's fun enough to implement, but is there a real-world use for this? I can't think of one. Usually if you want to remove elements from a list you'd use reject; having a list of indexes as an intermediary step seems like quite an unusual situation to me. Just because there's a positive counterpart doesn't mean we need a negative counterpart "for the sake of consistency".

#10 - 03/13/2020 08:26 PM - alex_golubenko (Alex Golubenko)

Dan0042 (Daniel DeLorme) wrote in #note-9:

> That implementation would return an empty array for [true,nil,nil].except_index(0)
> Probably something a bit more like this:
>
> ```
>   @@undef = Object.new #using Qundef in C
>   def except_index(*indexes)
>     result = dup
>     indexes.each do |ind|
>       if ind.is_a?(Range)
>         ind.each{ |i| result[i] = @@undef }
>       else
>         result[ind] = @@undef
>       end
>     end
>     result.reject!{ |e| e == @@undef }
>     result
>   end
> ```
>
> It's fun enough to implement, but is there a real-world use for this? I can't think of one. Usually if you want to remove elements from a list you'd use reject; having a list of indexes as an intermediary step seems like quite an unusual situation to me. Just because there's a positive counterpart doesn't mean we need a negative counterpart "for the sake of consistency".

Sure, the final implementation was:

```
  def except_index(*indicies)
    to_delete = Array.new(length)
```

```
    indicies.each do |ind|
      if ind.is_a?(Range)
        ind.each { |i| i > length ? break : to_delete[i] = true }
      else
        to_delete[ind] = true
      end
    end
    reject.with_index { |_, ind| to_delete[ind] }
  end
```

As I said in the first message, I proposed it mostly because I found that questions on StackOverflow about this realization helpful for many developers, so I thought mostly about it rather than just something opposite to values_at :)

I agree with you that this method would not the most popular but I also heard from developers, that they faced tricky challenges where this method could help. :)

#11 - 03/15/2020 03:37 PM - Dan0042 (Daniel DeLorme)

alex_golubenko (Alex Golubenko) wrote in #note-10:

> I also heard from developers, that they faced tricky challenges where this method could help. :)

Then it would be really nice if you could show these tricky challenges here; that would be the best argument in favor.