Ruby master - Feature #12134

Comparison between `true` and `false`

03/02/2016 10:56 AM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

There are some needs to sort elements depending on whether they satisfy certain condition expressed as a predicate. For example, to place prime numbers before others:

```
require "prime"
[7, 6, 5, 4, 3, 2, 1].sort_by{|e| Prime.prime?(e) ? 0 : 1} # => [7, 5, 3, 2, 6, 4, 1]
```

or to do such sort with the secondary condition to sort by the size:

```
[7, 6, 5, 4, 3, 2, 1].sort_by{|e| [Prime.prime?(e) ? 0 : 1, e]} # => [2, 3, 5, 7, 1, 4, 6]
```

Here, the temporal assignment of magic numbers 0 and 1 is ad hoc, but ordering between true and false makes sense. And given that there are if construction (which is unmarked case compared to the unless construction) and the ternary operator, in which the truthy branch is placed before the falsy branch, I think it makes sense to assume an inherent ordering of true being placed before false.

So I propose comparison between true and false:

```
true <=> false # => -1
false <=> true # => 1
```

Using this, the cases above can be written more directly as:

```
[7, 6, 5, 4, 3, 2, 1].sort_by{|e| Prime.prime?(e)} # => [7, 5, 3, 2, 6, 4, 1]
[7, 6, 5, 4, 3, 2, 1].sort_by{|e| [Prime.prime?(e), e]} # => [2, 3, 5, 7, 1, 4, 6]
```

Please do not confuse this with the common proposal to map booleans to integers, particularly true.to_i # => 1 and false.to_i # => 0. That is arbitrary, and does not make sense. In fact, my proposal goes against such proposal (under the proposal to map booleans, true.to_i <=> false.to_i translates to 1 <=> 0 # => 1, which goes against my proposal true <=> false # => 01).

History

#1 - 03/02/2016 11:16 AM - Hanmac (Hans Mackowiak)

for the first case you can use #partition

```
[7, 6, 5, 4, 3, 2, 1].partition{|e| Prime.prime?(e) }.flatten
# => [7, 5, 3, 2, 6, 4, 1]
```

#2 - 03/05/2016 02:42 AM - shan (Shannon Skipper)

Just for comparison purposes, here's an example of the latter with #partition:

```
[7, 6, 5, 4, 3, 2, 1].partition(&:prime?).flat_map(&:sort)
#=> [2, 3, 5, 7, 1, 4, 6]
```

#3 - 03/05/2016 04:45 AM - sawa (Tsuyoshi Sawada)

As another example, let us consider displaying items in an issue tracker. There is an array items of items, each of Issue class. Issue class has attributes done (true/false), id (Integer), subject (String), time (DateTime). Let us assume that the items are to be displayed as a table in an order specified by the user input. In one occasion, the sorting code can be written as:

```
items.sort_by{|item| [item.time, item.done, item.subject, item.id]}
```

#4 - 03/07/2016 04:53 AM - duerst (Martin Dürst)

Tsuyoshi Sawada wrote:

> Please do not confuse this with the common proposal to map booleans to integers, particularly true.to_i # => 1 and false.to_i # => 0. That is arbitrary, and does not make sense.

In absolute terms, it is arbitrary. But it's the most widely used mapping, and therefore would make much more sense than any other mapping.

> In fact, my proposal goes against such proposal (under the proposal to map booleans, true.to_i <=> false.to_i translates to 1 <=> 0 # => 1, which goes against my proposal true <=> false # => 01).

As far as I can remember, it's customary to define true being greater than false. I think this can be explained by the fact that true (something) is easily considered greater than false (nothing).

I think it's worth to check other programming languages. In Python, for example:

```
>>> True > False
True
```

Your specific example can still easily be written as:

```
[7, 6, 5, 4, 3, 2, 1].sort_by{|e| [not Prime.prime?(e), e]} # => [2, 3, 5, 7, 1, 4, 6]
```

#5 - 03/15/2016 08:50 AM - sawa (Tsuyoshi Sawada)

Martin Dürst wrote:

> Tsuyoshi Sawada wrote:
>
> > Please do not confuse this with the common proposal to map booleans to integers, particularly true.to_i # => 1 and false.to_i # => 0. That is arbitrary, and does not make sense.
>
> In absolute terms, it is arbitrary. But it's the most widely used mapping, and therefore would make much more sense than any other mapping.

I agree that it is the most common mapping, but I would say that that is an implementation detail (bit level). From the perspective of OOP, a user should be using true/false when they want to express logic, and not 1/0.

> As far as I can remember, it's customary to define true being greater than false. I think this can be explained by the fact that true (something) is easily considered greater than false (nothing).

That way of thinking is also arbitrary, I think. If you think of a birth of something, it goes from non-existence to existence, but if you think of the death/disappearance of something, it goes from existence to non-existence. Other than that, perhaps you are assuming the mapping true => 1 and false => 0, further interpreting the numbers as cardinality. But as I wrote above, I think the mapping itself is insignificant, and should not be considered at the level of OOP.

> I think it's worth to check other programming languages. In Python, for example:
>
> ```
> >>> True > False
> True
> ```

I think this comes from the mapping in mind, which I am claiming against.

> Your specific example can still easily be written as:
>
> ```
> [7, 6, 5, 4, 3, 2, 1].sort_by{|e| [not Prime.prime?(e), e]} # => [2, 3, 5, 7, 1, 4, 6]
> ```

Yes, that is one important point. Whichever way the comparison is defined, you should be able to reverse it by applying negation on it.

As others have shown, with my proposed order, we can get results similar to what other methods like partition do. If the order is decided the other way around, it would be confusing.