Let `exception` option in `Kernel#Complex`, `Kernel#Float`, `Kernel#Integer`, `Kernel#Rational` be falsy vs. truthy

07/05/2019 03:12 AM - sawa (Tsuyoshi Sawada)

Status: Closed
Priority: Normal
Assignee: 
Target version: 
ruby -v: 
Backport: 2.5: UNKNOWN, 2.6: UNKNOWN

Description
The exception option in Kernel#Complex, Kernel#Float, Kernel#Integer, and Kernel#Rational distinguishes false vs. other values.

```
Integer("z", exception: false) #=> nil
Integer("z", exception: nil) #=> ArgumentError: invalid value for Integer(): "z"
```

But in most other cases where a boolean notion is concerned (for example, the chomp option in Kernel#gets), the distinction is between falsy vs. truthy values.

I request the distinction to be falsy vs. truthy. In other words, I would like the value nil to work on the falsy side rather than the truthy side like this.

```
Integer("z", exception: false) #=> nil
Integer("z", exception: nil) #=> nil
```

Associated revisions
Revision 3e7d0021 - 07/11/2019 11:04 AM - nobu (Nobuyoshi Nakada)
Check exception flag as a bool [Bug #15987]

Revision c2723e59 - 07/11/2019 11:20 AM - nobu (Nobuyoshi Nakada)
Removed wrong argument in the fallback function [Bug #15987]

Revision f74e23af - 07/11/2019 11:21 AM - nobu (Nobuyoshi Nakada)
Fixed argument in the fallback function [Bug #15987]

Revision 8745fa2f - 07/11/2019 12:05 PM - nobu (Nobuyoshi Nakada)
Default to true when no exception flag [Bug #15987]

History
#1 - 07/05/2019 03:27 AM - sawa (Tsuyoshi Sawada)
- Subject changed from Let `exception` option in `Kernel#Complex`, `Kernel#Float`, `Kernel#Integer`, `Kernel#Rational` be falsy vs. truthy to Let boolean option (such as `exception` in `Kernel#Complex`, `Kernel#Float`, `Kernel#Integer`, `Kernel#Rational`) be falsy vs. truthy
- Description updated

#2 - 07/05/2019 06:02 AM - shevegen (Robert A. Heiler)
I don't have a huge preference either way, but this may be a design decision, so perhaps we can ask matz either way; also in how strong the concept of "falsy" and "truthy" is in ruby.
I assume there were design considerations for when matz added nil and false and not considered both to be equivalent (e. g. if some value is nil, you can use that to distinguish it from true or false).

I think we can find examples where nil being treated as false makes sense, but we may also consider cases where nil being false does not make a whole lot of sense, e. g. if that variable would lateron be initialized to become a String or Array. Perhaps that may have been a reason why you did limit the suggestion to only specific methods, e. g. chomp in Kernel#gets. For specific methods, it may be that it makes sense to only distinguish true/false boolean states.

Aside from the conceptual and design consideration, what impact would we be able to
observe if this would be changed? I have not thought through all the given methods yet. :)

#3 - 07/06/2019 02:00 PM - Eregon (Benoit Daloze)
I agree for methods taking an :exception keyword argument.
I would consider this a bug to treat nil as "want exception".
Do you have examples of other methods taking a boolean option and treating nil the same as true?
Those should probably be fixed too, but they might be hard to find.

#4 - 07/06/2019 04:45 PM - mame (Yusuke Endoh)
@sawa (Tsuyoshi Sawada) Could you write a code example in the description? It would be very helpful to make the developer meeting efficiently.

```
p Integer("z", exception: false) #=> nil
p Integer("z", exception: nil) #=> excepted: nil, actual: invalid value for Integer(): "z" (ArgumentError)
```

#5 - 07/10/2019 05:01 AM - sawa (Tsuyoshi Sawada)
- Description updated

#6 - 07/10/2019 07:41 AM - sawa (Tsuyoshi Sawada)
- Subject changed from Let boolean option (such as `exception` in `Kernel#Complex`, `Kernel#Float`, `Kernel#Integer`, `Kernel#Rational`) be falsy vs. truthy to Let `exception` option in `Kernel#Complex`, `Kernel#Float`, `Kernel#Integer`, `Kernel#Rational` be falsy vs. truthy
- Description updated

#7 - 07/11/2019 07:30 AM - matz (Yukihiro Matsumoto)
- Tracker changed from Feature to Bug
- Backport set to 2.5: UNKNOWN, 2.6: UNKNOWN

It's a bug. It should be fixed.
Matz.

#8 - 07/11/2019 08:50 AM - nobu (Nobuyoshi Nakada)
Note: The rdoc of Kernel#Integer says:
```
Passing nil raises a TypeError, while passing a String that
does not conform with numeric representation raises an ArgumentError.
```

and states:
```
This behavior can be altered by passing exception: false, in this case a not convertible value will return nil.
```

So the current behavior is as documented, at least.
And this seems coming from IO#read_nonblock and so on.

#9 - 07/11/2019 09:17 AM - sawa (Tsuyoshi Sawada)
nobu (Nobuyoshi Nakada) wrote:
```
this seems coming from IO#read_nonblock and so on.
```
I initially had a feeling that I had seen this opposition between false vs. other values somewhere else, but had forgotten where. Now that nobu has pointed it, I would like to request these related keywords be turned to falsy vs. truthy as well.

#10 - 07/11/2019 11:15 AM - nobu (Nobuyoshi Nakada)
- Status changed from Open to Closed

Applied in changeset git|3e7d002118a92fad5934e11c75b6e6768a1476c1b.

Check exception flag as a bool [Bug #15987]
It doesn't feel good for me to allow arbitrary values other than true as truthy. nil as falsy is not so weird...

Options are:

- accept normal truthy/falsey value for boolean (using RTEST())
- accept true/false for boolean; exception for everything else

And we should keep consistency.

Matz.

I think nil means default behavior.

That does not hold even if you just look at the Kernel methods:

exit(status=true)
Kernel::exit(status=true)
Process::exit(status=true)
exit!(status=false)
gets(sep=/ [, getline_args])
rand(max=0)
readline(sep=/)
readlines(sep=/)

These methods have default values other than nil.

Both sound fine to me.

nobu implemented the second option.

We can try to keep consistency in the future, but I think we can't easily change existing methods taking a truthy/falsey argument for compatibility.

I think nil means default behavior.
That does not hold even if you just look at the Kernel methods:

As for optional parameters, there are some inconsistencies.

```
exit(status=true)
Kernel::exit(status=true)
Process::exit(status=true)
exit!(status=false)
gets(sep=$/, getline_args))
rand(max=0)
readline(sep=$/)
readlines(sep=$/)
```

These methods have default values other than nil.

And other methods fall back nil to the default value, e.g., File.open.

#17 - 07/13/2019 03:28 AM - sawa (Tsuyoshi Sawada)

nobu (Nobuyoshi Nakada) wrote:

As for optional parameters, there are some inconsistencies.

... And other methods fall back nil to the default value, e.g., File.open.

I came up with another alternative.

Since this is about suppressing an exception, I think the concept is close to the optional second argument/block of Hash#fetch. One is keyword argument and the other is positional argument, but that does not mean that the values should behave differently. I think we can imitate it.

That is, when the explicit exception option is given, then that value should be returned instead of raising an error when the string cannot be parsed correctly. This would be a breaking change, but I do not think it will break much existing code because most use cases so far probably only use the exception option with the value false, in which case it currently returns nil, and my proposal would change that to return false, but I do not think that is a big deal.

#18 - 07/13/2019 10:18 AM - Eregon (Benoit Daloze)

This issue is closed, it's probably best to discuss on a new one for different ideas.

sawa (Tsuyoshi Sawada) wrote:

but I do not think that is a big deal.

IMHO it is a big deal, Integer() should return either an Integer or nil, but never false or any other kind of value. nil means "absence of value", false doesn't have the same meaning.

I believe everyone understands the exception: keyword should be given some kind of boolean value, using the value for anything else than deciding whether to raise exceptions would be very surprising.