Ruby master - Bug #13064

Inconsistent behavior with `next` inside `begin`/`end` across different implementations.

12/23/2016 04:07 PM - jwmittag (Jörg W Mittag)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.4.0dev (2016-12-22 trunk 57151) [x86_64-darwin16]	**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

@Kalsan over on StackOverflow observed inconsistent behavior with next inside begin/end across different implementations. In particular, Rubinius, JRuby, and MRuby all behave the same while YARV behaves differently. In addition, running the same code inside IRb behaves differently than running it as a script.

Usually, YARV is the gold standard, and if some behavior differs from YARV, YARV is right and the other implementation is wrong. However, in this case, not only do *all other* implementations disagree with YARV, but in particular, MRuby/Rite, the implementation written and maintained by matz himself disagrees with YARV.

So, what is the correct behavior for this (admittedly) non-sensical code?

Here is the behavior across different versions of YARV and different implementations:

- YARV 2.2.0 (the build shipping with macOS)

```
# ruby -v -W -e 'begin; begin puts 1; next end; puts 2 end'
ruby 2.0.0p648 (2015-12-16 revision 53162) [universal.x86_64-darwin16]
-e:1: warning: statement not reached
-e:1: Invalid next
-e: compile error (SyntaxError)
```

- YARV 2.3.1 (the version JRuby claims to be compatible with at the moment), 2.3.3, 2.4.0-preview3, 2.4.0-dev (current SVN trunk as of yesterday):

```
# ruby -v -W -e 'begin; begin puts 1; next end; puts 2 end'
ruby 2.4.0dev (2016-12-22 trunk 57151) [x86_64-darwin16]
-e:1: warning: statement not reached
-e: -e:1: Invalid next (SyntaxError)
```

- Rubinius 3.69

```
# rbx -v -W -e 'begin; begin puts 1; next end; puts 2 end'
rubinius 3.69 (2.3.1 a57071c6 2016-11-17 3.8.1) [x86_64-darwin15.6.0]
1
                main # Rubinius::Loader at core/loader.rb:860
               evals # Rubinius::Loader at core/loader.rb:646
                eval # Kernel(Rubinius::Loader) at core/kernel.rb:1130
    call_on_instance # Rubinius::BlockEnvironment at core/block_environment.rb:147
   { } in __script__ # Object at -e:1
          jump_error . Rubinius at core/rubinius.rb:279

invalid context for 'next' (LocalJumpError)

An exception occurred evaluating command line code
```

- JRuby 9.1.6.0 (the latest release)

```
# jruby -v -W -e 'begin; begin puts 1; next end; puts 2 end'
jruby 9.1.6.0 (2.3.1) 2016-11-09 0150a76 Java HotSpot(TM) 64-Bit Server VM 25.102-b14 on 1.8.0
_102-b14 +jit [darwin-x86_64]
1
LocalJumpError: unexpected next
```

```
  <main> at -e:1
```

- MRuby 1.2.0 (the minimal ISO compliant Ruby implementation written by matz himself)

```
# mruby -v -e 'begin; begin puts 1; next end; puts 2 end'
mruby 1.2.0 (2015-11-17)
00001 NODE_SCOPE:
00001   NODE_BEGIN:
00001     NODE_BEGIN:
00001       NODE_BEGIN:
00001         NODE_CALL:
00001           NODE_SELF
00001           method='puts' (383)
00001           args:
00001             NODE_INT 1 base 10
00001         NODE_NEXT:
00001       NODE_CALL:
00001         NODE_SELF
00001         method='puts' (383)
00001         args:
00001           NODE_INT 2 base 10
irep 0x7fe0e3c1b630 nregs=4 nlocals=1 pools=1 syms=1 reps=0
file: -e
    1 000 OP_LOADSELF   R1
    1 001 OP_LOADI  R2  1
    1 002 OP_SEND   R1  :puts   1
    1 003 OP_ERR    "unexpected next"
    1 004 OP_LOADSELF   R1
    1 005 OP_LOADI  R2  2
    1 006 OP_SEND   R1  :puts   1
    1 007 OP_STOP

1
trace:
    [0] -e:1
LocalJumpError: unexpected next
```

What is most interesting is that MRuby, JRuby and Rubinius actually agree on the behavior, but differ from YARV. Either YARV or all the other ones are wrong. I cannot say which ones, though.

History

#1 - 12/24/2016 10:05 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

It's a REPL implementation dependent (irb uses eval() inside). Try comparing them using file execution.

Matz.

#2 - 12/27/2016 12:51 AM - jwmittag (Jörg W Mittag)

Yukihiro Matsumoto wrote:

> It's a REPL implementation dependent (irb uses eval() inside). Try comparing them using file execution.

I did, the samples I posted contain both script execution (ruby -e) and IRb interactive sessions. There is a behavioral difference between YARV and all other implementations: YARV issues a warning "statement not reached", the others don't. YARV raises a SyntaxError "invalid next", the others raise a LocalJumpError "unexpected next". YARV doesn't execute anything (it's a syntax error, after all), the others execute the first puts.

Normally, "YARV is always right", because it is developed with your (matz's) involvement and you are the creator of Ruby, so you are always right. But in this particular case, *all* (or at least a significant number) other implementations perfectly agree with each other and disagree with YARV, *and* one of those implementations that disagree with YARV is MRuby, written by you, and intended to be standards-compliant.

There are a couple of possibilities:

1. YARV is right, all the others are wrong.
2. The others are right, YARV is wrong.

3. All are wrong, the correct behavior is something else entirely.
4. It is implementation-defined behavior, all are right.
5. It is undefined behavior, anything can happen.
6. The code is stupid, nobody would ever write something like that! :-D (Well, that's probably true, but nonetheless, it would be nice to have definite ruling.)

Note: I am removing the IRb samples from the report, they are only distracting from the issue.

#3 - 12/27/2016 12:55 AM - jwmittag (Jörg W Mittag)

- Description updated

#4 - 12/27/2016 02:03 AM - nobu (Nobuyoshi Nakada)

It's 4, I guess.

#5 - 01/19/2017 07:44 AM - ko1 (Koichi Sasada)

1!

#6 - 01/19/2017 07:56 AM - matz (Yukihiro Matsumoto)

mruby will be compatible with YARV soon.

Matz.

#7 - 01/20/2017 04:22 AM - shyouhei (Shyouhei Urabe)

We looked at this issue at yesterday's developer meeting.

While Ko1 and Matz already replied that the current MRI behaviour is their intension, I confirmed that ISO/IEC 30170 says nothing about this situation. So this is at least not "someone did something wrong" situation.

I think both behaviour sounds valid so I bet it's 4.

#8 - 07/25/2019 11:41 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Feedback to Closed