Ruby - Feature #10930

Allow splat operator to work for string interpolation

03/03/2015 02:41 PM - danielpclark (Daniel P. Clark)

Status:	Feedback
Priority:	Normal
Assignee:	
Target version:	

Description

Currently when you use the splat operator in a method it pulls the items out of the array for method parameters.

```
def foo(a,b,c)
  puts "#{a}, #{b}, #{c}"
end

bar = [1,2,3]
foo(*bar)
# => "1, 2, 3"
```

So I would expect to be able to do "#{*bar}" and get either "1, 2, 3" or "1,2,3". But when attempting this I get.

```
baz = [1,2,3]

"#{*baz}"

# SyntaxError: (irb):53: syntax error, unexpected tSTRING_DEND, expecting '='
# "#{*baz}"
#         ^
# (irb):53: unterminated string meets end of file

"#{*[1,2,3]}"

# SyntaxError: (irb):54: syntax error, unexpected tSTRING_DEND, expecting :: or '[' or '.'
# "#{*[1,2,3]}"
#             ^
# (irb):54: unterminated string meets end of file
```

This doesn't work on any of the Ruby versions available 1.8 through 2.2.1. They each produce the same error.

I propose allowing the splat operator within string interpolation to work the same as [1,2,3].join(',')

```
fiz = [1,2,3]

"#{*fiz}"
# => "1,2,3"

"#{*[1,2,3]}"
# => "1,2,3"
```

History

#1 - 03/04/2015 02:56 AM - matz (Yukihiro Matsumoto)

- Subject changed from Allow splat ooperator to work for string interpolation to Allow splat operator to work for string interpolation

- Status changed from Open to Feedback

I have no idea why you need this. Does this have common use-case?
For me, "#{*[1,2,3]}" to produce "1, 2, 3" is kinda arbitrary.

Matz.

#2 - 03/04/2015 04:01 AM - danielpclark (Daniel P. Clark)

The behavior of the splat operator as used in methods is like removing [ ] from an array to use as parameters.

```
eval("xaz = [1,2,3]; def biz(a,b,c) puts a, b, c; end; biz(*xaz)")
# 1
# 2
# 3
# => nil
```

So intuitively it would make sense that it would remove them for interpolation from a string.

```
zax = [1,2,3]
# => [1, 2, 3]

eval("def zib(a,b,c) puts a, b, c; end; zib(#{*zax})")
# SyntaxError: (irb):23: syntax error, unexpected tSTRING_DEND, expecting '='
# eval("def zib(a,b,c) puts a, b, c; end; zib(#{*zax})")
#                                                 ^
# (irb):23: unterminated string meets end of file
```

It seems odd to only work this way in the method argument call when it can be used the same way.

```
eval("def zab(a,b,c) puts a, b, c; end; zab(*#{zax})")
# 1
# 2
# 3
# => nil
```

It's not so much a "need" as it behaves differently than expected in my mind. When I explain the splat operator to people I say that it brakes the values out of an Array. So just thinking in this way the splat turns my_method(*[1,2,3]) into my_method(1,2,3) then it would make sense to do the same for "#{*[1,2,3]}" into "1,2,3".

#3 - 03/04/2015 08:58 PM - phluid61 (Matthew Kerwin)

On 4 March 2015 at 14:01, 6ftdan@gmail.com wrote:

> Issue #10930 has been updated by Daniel P. Clark.
>
> The behavior of the splat operator as used in methods is like removing [ ]
> from and array to use as parameters.
>
> So intuitively it would make sense that it would remove them for
> interpolation from a string.

S
e are you also requesting that "#{1, 2, 3}" be considered valid? Because that's how I would interpret "#{*[1,2,3]}" if I saw it in code.

I'm strongly opposed to it ever arbitrarily injecting commas, especially since Array#to_s joins without any delimiter.

--
Matthew Kerwin
http://matthew.kerwin.net.au/

#4 - 03/04/2015 09:04 PM - recursive-madman (Recursive Madman)

Matthew Kerwin wrote:

> I'm strongly opposed to it ever arbitrarily injecting commas, especially
> since Array#to_s joins without any delimiter.

It doesn't actually:

```
2.2.0 :001 > puts %w(a b c).to_s
["a", "b", "c"]
```

join joins without a comma:

```
2.2.0 :002 > puts %w(a b c).join
abc
```

Also, I agree with your point.

#5 - 03/04/2015 09:28 PM - phluid61 (Matthew Kerwin)

On 5 March 2015 at 07:04, recursive.madman@gmx.de wrote:

> Issue #10930 has been updated by Recursive Madman.
>
> Matthew Kerwin wrote:
>
> > I'm strongly opposed to it ever arbitrarily injecting commas, especially since Array#to_s joins without any delimiter.
>
> It doesn't actually:
>
> ```
> 2.2.0 :001 > puts %w(a b c).to_s
> ["a", "b", "c"]
> ```

Oh, wow, yeah that changed in 1.9. I must have been burned badly by 1.8 to carry that baggage this long. :)

--
Matthew Kerwin
http://matthew.kerwin.net.au/

#6 - 03/05/2015 03:20 AM - danielpclark (Daniel P. Clark)

Matthew Kerwin wrote:

> So are you also requesting that "#{1, 2, 3}" be considered valid? Because that's how I would interpret "#{*[1,2,3]}" if I saw it in code.

```
"#{1,2,3}"
# SyntaxError: (irb):1: syntax error, unexpected ',', expecting tSTRING_DEND
# "#{1,2,3}"
#       ^
```

I see your point. Comma seperated items themselves don't currently work in interpolation. So I guess this suggestion isn't as valid as I thought it was. I was just thinking of the same behavior "as a string" and not about the step in between.

#7 - 12/05/2015 04:53 AM - danielpclark (Daniel P. Clark)

To better demonstrate the pain point here.

```
class Example
  def method_missing m, *a
    puts "Method #{m} called with arguments ", *a
  end
end

Example.new.foo :bar, :baz
#Method foo called with arguments
#bar
#baz

class Example
  def method_missing m, *a
    puts "Method #{m} called with arguments #{a}"
  end
end

Example.new.foo :bar, :baz
#Method foo called with arguments [:bar, :baz]
```

Neither of these are the desired behavior. It ends up needing to be implemented like this.

```
class Example
  def method_missing m, *a
    puts "Method #{m} called with arguments #{a.to_s[1..-2]}"
  end
end

Example.new.foo :bar, :baz
#Method foo called with arguments :bar, :baz
```

It's not a big deal. Just an inconvenience .

```
class String
  def crop(edge=1)
    self[edge..-(edge+1)]
  end
end

class Array
  def to_str
    to_s.crop
  end
end

class Example
  def method_missing m, *a
    puts "Method #{m} called with arguments #{a}"
  end
end

Example.new.foo :bar, :baz
#Method foo called with arguments :bar, :baz
```

This ^ would make me happy :-)

Just a note... interpolation seems to call to_s and not to_str on an Array. Shouldn't to_str be the first called method on implicit conversion of an Array to a String?