Ruby - Bug #10837

spatting a single element array produces an array instead of a single value for return and next

02/08/2015 12:01 AM - bughit (bug hit)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.2.0p0 (2014-12-25 revision 49005) [x86_64-linux]	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

```
irb(main):013:0> ->{return *[1]}.()
=> [1]
irb(main):014:0> ->{next *[1]}.()
=> [1]
```

---

*[x] should mean x as it already does for arguments

History

#1 - 02/08/2015 01:48 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Rejected

It's similar to return *[1, 2].

#2 - 02/08/2015 02:02 AM - bughit (bug hit)

Nobuyoshi Nakada wrote:

> It's similar to return *[1, 2].

No it's not similar,

return *[1, 2] means return 1, 2

similar would be:

return *[1] means return 1

in general, splatting in a context that takes a coma separated list of items (method args, rescue, return, next, multiple assignment) is supposed to destructure the array. The array should be gone, only the elemnts remain, so returning the [1] is wrong.

#3 - 02/08/2015 02:25 AM - bughit (bug hit)

bug hit wrote:

> Nobuyoshi Nakada wrote:
>
> > It's similar to return *[1, 2].
>
> No it's not similar,
>
> return *[1, 2] means return 1, 2
>
> similar would be:
>
> return *[1] means return 1
>
> in general, splatting in a context that takes a coma separated list of items (method args, rescue, return, next, multiple assignment) is supposed to destructure the array. The array should be gone, only the elemnts remain, so returning the [1] is wrong.

some examples

method_call *[a] is method_call a as expected
rescue *[a] is rescue a as expected
[*[a]] is [a] as expected
b = *[a] is b = [a] why?
(next|return) *[a] is (next|return) [a] why?

the intuitive, expected and logical behavior would be for *array to consistently destructure, when used in a coma separated list context

#4 - 02/09/2015 11:18 PM - bughit (bug hit)

bug hit wrote:

> bug hit wrote:
>
> > Nobuyoshi Nakada wrote:
> >
> > > It's similar to return *[1, 2].
> >
> > No it's not similar,
> >
> > return *[1, 2] means return 1, 2
> >
> > similar would be:
> >
> > return *[1] means return 1
> >
> > in general, splatting in a context that takes a coma separated list of items (method args, rescue, return, next, multiple assignment) is supposed to destructure the array. The array should be gone, only the elemnts remain, so returning the [1] is wrong.
>
> some examples
>
> method_call *[a] is method_call a as expected
> rescue *[a] is rescue a as expected
> [*[a]] is [a] as expected
> b = *[a] is b = [a] why?
> (next|return) *[a] is (next|return) [a] why?
>
> the intuitive, expected and logical behavior would be for *array to consistently destructure, when used in a coma separated list context

please explain

#5 - 02/10/2015 03:25 PM - marcandre (Marc-Andre Lafortune)

bug hit wrote:

> b = *[a] is b = [a] why?
> (next|return) *[a] is (next|return) [a] why?

I agree, this can be surprising.

The reason for the behavior is that return 1, 2, strictly speaking, shouldn't be valid Ruby as you can only return one value. Instead of forbidding it, return 1, 2 is automatically converted to return [1, 2]. They are equivalent. So return *array is converted to return [*array], and that holds even in the cases where array contains one or no element (or isn't an array).

The same can be said for next and =.

HTH

#6 - 02/10/2015 04:23 PM - bughit (bug hit)

Marc-Andre Lafortune wrote:

> bug hit wrote:
>
> > b = *[a] is b = [a] why?
> > (next|return) *[a] is (next|return) [a] why?
>
> I agree, this can be surprising.
>
> The reason for the behavior is that return 1, 2, strictly speaking, shouldn't be valid Ruby as you can only return one value. Instead of forbidding it,

> return 1, 2 is automatically converted to return [1, 2]. They are equivalent. So return *array is converted to return [*array], and that holds even in the cases where array contains one or no element (or isn't an array).
>
> The same can be said for next and =.
>
> HTH

So it seems it's an implementation artifact. Would it not be better if semantics of splatting were consistent, i.e. rvalue splat would always destructure the array?

#7 - 02/25/2015 04:20 PM - bughit (bug hit)

Nobuyoshi Nakada wrote:

> It's similar to return *[1, 2].

Please clarify your position, is this about preserving compatibility, or do you really disagree that conceptually an rvalue splat should eliminate the array?

#8 - 04/01/2015 06:12 PM - bughit (bug hit)

please explain

#9 - 07/18/2016 12:47 PM - najamelan (Naja Melan)

bug hit wrote:

> Nobuyoshi Nakada wrote:
>
> > It's similar to return *[1, 2].
>
> Please clarify your position, is this about preserving compatibility, or do you really disagree that conceptually an rvalue splat should eliminate the array?

Actually this is really useful. Whenever a method accepts a something or an array of somethings you can use the splat operator to make it an array if it's not already. Now you can operate on it as an array.

```
def initialize( something )

   @something = *something

end
```

If it didn't do that, you would have to write:

```
@something = something.kind_of?( Array ) ? something : [something]
```

#10 - 07/18/2016 05:50 PM - bughit (bug hit)

Naja Melan wrote:

> Actually this is really useful. Whenever a method accepts a something or an array of somethings you can use the splat operator to make it an array if it's not already.

This is a bad hack that produces obfuscated code and does not work properly, if you splat a hash you will end up with an array of arrays. What you want is something like Array.wrap from rails, which both communicates intention clearly and works universally.

Splatting an array should consistently destructure it.

#11 - 07/18/2016 05:58 PM - najamelan (Naja Melan)

@bug hit

My apologies, it seems I'm mistaken. The splat operator does not just seem to coerce into array for all types:

```
a = { a: 1 }

p [a]   # [{:a=>1}]
p *a    # [:a, 1]
p [*a]  # [[:a, 1]]
```

It seems it calls #to_a on classes that implement this. I'm sorry if I create confusion in the ongoing discussion.

#12 - 07/18/2016 06:06 PM - bughit (bug hit)

Naja Melan wrote:

> @bug hit
>
> My apologies, it seems I'm mistaken. The splat operator does not just seem to coerce into array for all types:
>
> ```
> a = { a: 1 }
>
> p [a]   # [{:a=>1}]
> p *a    # [:a, 1]
> p [*a]  # [[:a, 1]]
> ```
>
> It seems it calls #to_a on classes that implement this. I'm sorry if I create confusion in the ongoing discussion.

Also when it's already an array (a = *array), what you are asking ruby to do is first destructure the array, then coalesce it back into an array, essentially making a shallow copy, whereas what you actually want is to leave it alone. Array.wrap leaves it alone.