Ruby - Feature #8563

Instance variable arguments

06/23/2013 04:30 AM - sawa (Tsuyoshi Sawada)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

Often times, people want to assign given arguments to instance variables, especially inside the method initialize:

```
def initialize foo, bar, buz
  @foo, @bar, @buz = foo, bar, buz
  ...
end
```

I propose to let method definition take instance variables as arguments so that:

```
def initialize @foo, @bar, @buz
  ...
end
```

would be equivalent as above.

Related issues:

Is duplicate of Ruby - Feature #5825: Sweet instance var assignment in the ob...	**Assigned**
Has duplicate Ruby - Feature #12578: Instance Variables Assigned In parameter...	**Rejected**
Has duplicate Ruby - Feature #12820: Shorter syntax for assigning a method ar...	**Rejected**

History

#1 - 06/23/2013 09:41 AM - nobu (Nobuyoshi Nakada)

- Category set to syntax

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

- Target version set to 3.0

#2 - 06/23/2013 03:24 PM - phluid61 (Matthew Kerwin)

Question: would this be valid?

```
def foo(@foo=@foo) end
```

Asking here, because #5825 as written only talks about initialize method, and the above construct wouldn't make sense there.

#3 - 06/23/2013 08:34 PM - matz (Yukihiro Matsumoto)

From my POV:

```
def initialize(@foo, @bar)
end
```

does not express intention of instance variable initialization. I'd rather add a method like

```
define_attr_initialize(:foo, :bar)
```

to define a method of

```
def initialize(foo, bar)
  @foo = foo
  @bar = bar
end
```

Matz.

#4 - 06/23/2013 10:21 PM - sawa (Tsuyoshi Sawada)

It could also be used besides initialize:

```
def update_something foo
   do_update_something(@foo = foo)
   ...
end
```

would become

```
def update_something @foo
   do_update_something(@foo)
   ...
end
```

#5 - 06/23/2013 10:37 PM - nobu (Nobuyoshi Nakada)

phluid61 (Matthew Kerwin) wrote:

> Question: would this be valid?
>
> ```
> def foo(@foo=@foo) end
> ```

In generic,

```
foo = foo
```

is valid always.

#6 - 06/24/2013 01:29 AM - headius (Charles Nutter)

Worth pointing out that blocks used to support this:

```
1.times { |@foo| ... }
```

Basically, it supported anything you can have on the LHS of a normal assignment:

```
foo.bar { |a, @b, @@c, D, e.val, f[0]| ... }
```

I believe it was taken out in 1.9 because it made argument processing a lot more complicated, but then 1.9 added the ability to do multiple-assignment grouping, default values, and other masgn features to all argument lists anyway. It doesn't seem like it would be too terribly complicated to add this back in, but I wonder about the OTHER reasons that non-local variable assignment was removed from argument lists in the first place.

As for the utility of the feature, I'd like it but I can live without it. It does seem rather un-Ruby to have to declare locals and do the assignment when all you want is to set an instance variable...especially when you have a lot of instance variables.

Note also that for a trivial initialize, where the only line is a multiple-assignment to set instance variables, every initialize call pays the cost of creating an Array for the masgn (see my rejected request in https://bugs.ruby-lang.org/issues/6668).

```
def initialize(a, b, c, d, e)
  @a, @b, @c, @d, @e = a, b, c, d, e # creates transient array every time
end
```

#7 - 06/24/2013 01:30 AM - Anonymous

@matz (Yukihiro Matsumoto): If define_attr_initialize is an option, then there is a question of named / ordered qualifier, either as:

```
define_attr_initialize :foo, :bar, as: :named
define_attr_initialize :baz, :quux, as: :ordered
```

or as (and I like this second option better):

```
attr_init_named foo: nil, bar: nil
attr_init_ordered :baz, :quux
```

Both of these would obviously stand for (using @sawa's notation):

```
def initialize( @baz, @quux, @foo: nil, @bar: nil )
      ...
```

Besides that, I feel that attr_reader, attr_writer, attr_accessor could use:

```
attr_reader :foo, :bar, init: :named
```

```
attr_reader :baz, :quux, init: :ordered
```

Which also brings to my mind, that now that we have named args firmly in place, following
syntactic flavors of Module#attr_... beg to exist:

```
attr_reader foo: 42, bar: 43  # specifying starting values explicitly
```

#8 - 06/24/2013 08:49 AM - phluid61 (Matthew Kerwin)

boris_stitnicky (Boris Stitnicky) wrote:

> Which also brings to my mind, that now that we have named args firmly in place, following
> syntactic flavors of Module#attr_... beg to exist:
>
> ```
> attr_reader foo: 42, bar: 43  # specifying starting values explicitly
> ```

If you propose this as a feature, I will +1 it. Also I have some questions about it which probably should not pollute the current feature request.

#9 - 06/24/2013 01:56 PM - Anonymous

> If you propose this as a feature, I will +1 it. Also I have some questions about it which probably should not pollute the current feature request.

I have proposed is as #8564.

#10 - 01/27/2016 09:47 AM - nobu (Nobuyoshi Nakada)

- Description updated

#11 - 07/10/2016 12:59 PM - nobu (Nobuyoshi Nakada)

- Has duplicate Feature #12578: Instance Variables Assigned In parameters (ala Crystal?) added

#12 - 07/19/2016 09:46 AM - Eregon (Benoit Daloze)

Anonymous wrote:

> @matz (Yukihiro Matsumoto): If define_attr_initialize is an option, then there is a question of named / ordered qualifier, either as:
>
> ```
> define_attr_initialize :foo, :bar, as: :named
> define_attr_initialize :baz, :quux, as: :ordered
> ```

You could have simply:

```
define_attr_initialize :baz, :quux, foo: nil, bar: nil
```

But this starts to look much like a macro to me.

One problem with the proposed define_attr_initialize is once some extra behavior needs to be added to initialize,
there is no choice but to desugar everything (write ivar assignments manually).
The original proposition does not have this issue.

IMHO many constructors usually need some custom behavior after some time,
and so paying the "cost" upfront of doing manual assignments is worth it in many cases.

#13 - 07/19/2016 01:43 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Even if define_attr_initialize would accept a block to enhance the initializer besides initializing instance variables it doesn't support default arguments. I'd certainly prefer the original proposal which I find more useful. It's not obvious what array.each(&:to_i) does either but people get used to it because it's handy. I think it's the same about this proposal although it's quite familiar for people using this feature in other languages like CoffeeScript and Crystal among others.

#14 - 07/19/2016 01:44 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Also it doesn't handle variable arguments, extra options that shouldn't be assigned to instance variables or &block.

#15 - 10/08/2016 02:38 AM - nobu (Nobuyoshi Nakada)

- Has duplicate Feature #12820: Shorter syntax for assigning a method argument to an instance variable added

#16 - 11/27/2017 02:42 PM - marcandre (Marc-Andre Lafortune)

matz (Yukihiro Matsumoto) wrote:

> Arguments are giving names to passed values, which is different from attribute (instance variables) initialization. I think they should be separated clearly.

This may be a valid theoretical distinction, but what does this change in practice?

I feel there would be about no additional cognitive load if this was accepted. It is most probably the most requested feature

This is not surprising, since a majority of initialize with arguments is doing that!

> And for most of the cases, only initialize needs this kind of initialization.

Indeed. This doesn't change the case that a majority of initialize methods needs this kind of initialization.

Here's an example Rails app (which I didn't write):
https://github.com/WikiEducationFoundation/WikiEduDashboard/search?utf8=%E2%9C%93&q=def+initialize&type=
A gem I'm using a lot these days: https://github.com/whitequark/parser/search?utf8=%E2%9C%93&q=def+initialize&type=

Since keyword arguments, I feel that's even more true, since Struct is no longer as useful.

> My idea is https://bugs.ruby-lang.org/issues/8563#note-3

With all due respect, this is not a good idea and it can't solve the issue. As was pointed out, there are many more cases that it doesn't handle:

- keyword arguments (that's absolutely major!)
- defaults
- dealing with other arguments
- doing extra code

I'd also mention the additional cognitive load involved.

Note that it has always been trivial to implement define_attr_initialize in pure Ruby. I've never seen such code though.

Take the top 100 gems, a few web apps, make an inventory of the initialize with parameters. I'm betting that the vast majority of these methods would be simplified by allowing instance variable arguments. I doubt that define_attr_initialize would help any but a small minority.

#17 - 11/29/2017 06:25 AM - ko1 (Koichi Sasada)

Not so serious idea.

```
class Binding
  def set_attributes(attrs = self.local_variables)
    attrs.each{|attr|
      self.receiver.instance_variable_set(:"@#{attr}", self.local_variable_get(attr))
    }
  end
end

class C
  def initialize a, b
    binding.set_attributes
  end
end

p C.new(1, 2) #=> #<C:0x000001f30a8b95d0 @a=1, @b=2>
```

#18 - 11/29/2017 06:26 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

My opinion has not been changed from https://bugs.ruby-lang.org/issues/8563#note-3
I am strongly against code like

```
def initialize(@foo, @bar)
end
```

I don't care about define_attr_initialize not being "the solution".

Matz.

#19 - 11/29/2017 06:27 AM - ko1 (Koichi Sasada)

Oops.

```
class Binding
  def set_attributes(attrs = self.eval('method(__method__).parameters.map{|t, v| v}'))
    attrs.each{|attr|
      self.receiver.instance_variable_set(:"@#{attr}", self.local_variable_get(attr))
    }
  end
end

class C
  def initialize a, b
    x = y = nil
    binding.set_attributes
  end
end

p C.new(1, 2) #=> #<C:0x000001f30a8b95d0 @a=1, @b=2>
```