# Ruby trunk - Feature #5825

## Sweet instance var assignment in the object initializer

12/30/2011 10:49 PM - goshakkk (Gosha Arinich)

| | | |
|---|---|---|
| **Status:** | Assigned | |
| **Priority:** | Normal | |
| **Assignee:** | matz (Yukihiro Matsumoto) | |
| **Target version:** | Next Major | |

**Description**

I'm very excited about this feature in CoffeeScript, and think it might be a nice-to-have thing in Ruby 2.0.

That's how I think it would look like:

```
class Me
  def initialize(@name, @age, @location); end
end
```

So we can declare @variables in the initializer method parameters definition to avoid assigning instance variables from method arguments by hand, like:

```
class Me
  def initialize(name, age, location)
    @name = name
    @age = age
    @location = location
  end
end
```

Want to hear what do you guys think, does that feature worth being included in 2.0?

**Related issues:**

| | |
|---|---|
| Related to Ruby trunk - Feature #15192: Introduce a new "shortcut assigning" ... | **Open** |
| Has duplicate Ruby trunk - Feature #8563: Instance variable arguments | **Rejected** |
| Has duplicate Ruby trunk - Feature #12023: Allow ivars to be used as method a... | **Open** |
| Has duplicate Ruby trunk - Feature #12578: Instance Variables Assigned In par... | **Rejected** |
| Has duplicate Ruby trunk - Feature #12820: Shorter syntax for assigning a met... | **Rejected** |

### History

**#1 - 12/30/2011 11:07 PM - lisovskyvlad (Vlad Lisosvky)**

I like it. No stupid assigns.

**#2 - 12/31/2011 02:20 AM - rue (Eero Saynatkari)**

Would be nice, and should be able to coexist with normal parameters:

```
def foo(bar, @baz, quux = @moomin)
  …
end
```

And so on. What about splat- and block arguments? It gets a little ugly:

```
def foo(bar, *@baz, &@quux)
  …
end
```

**#3 - 12/31/2011 02:25 AM - rosenfeld (Rodrigo Rosenfeld Rosas)**

+1 - too common use case

**#4 - 12/31/2011 03:26 PM - shyouhei (Shyouhei Urabe)**

I liked this 1.8-specific trick:

```
define_method(:intialize){|@foo, @bar|}
```

but it was abondoned for any reason.

**#5 - 01/02/2012 03:45 AM - andhapp (Anuj Dutta)**

+1. Common use case.

**#6 - 01/02/2012 08:40 AM - alexeymuranov (Alexey Muranov)**

+1, why only initialize?

**#7 - 01/02/2012 05:41 PM - Eregon (Benoit Daloze)**

I think most of the time you need to parse or check your arguments, in which case this syntax would not be practical.
Otherwise, you could use Struct to avoid the duplication:

```
class Me < Struct.new(:name, :age, :location)
end
```

**#8 - 01/03/2012 01:54 AM - goshakkk (Gosha Arinich)**

Alexey Muranov wrote:

> +1, why only initialize?

It's just too common case. It could work for other methods as well.

**#9 - 01/03/2012 07:38 AM - shevegen (Robert A. Heiler)**

Is this even possible with the Parser?

It would eliminate a few lines of code.

Also, I am not sure if this violates the principle of least matz surprise.

If I initially see

```
def initialize(@name, @age, @location)
```

I wonder a bit, because it does not feel consistent.

Perhaps it would be different if some kind of attr* could be
used.

```
attr_initialize :name, :age, :location
```

And then the above could work. Where the name would work just
similar to an attr_writer, but different in that it assumes
default values passed to initialize to automatically go towards
those instance variables (the order must be the same of course)

**#10 - 01/05/2012 04:24 PM - naruse (Yui NARUSE)**

*- Status changed from Open to Assigned*

*- Assignee set to matz (Yukihiro Matsumoto)*

**#11 - 02/25/2012 02:25 PM - ko1 (Koichi Sasada)**

*- Target version set to 2.0.0*

if my memory serves me right, matz dislikes such a style of arguments.
Matz, could you explain the reason again?

I know some people like this style.

**#12 - 02/25/2012 07:12 PM - trans (Thomas Sawyer)**

I think it's really not such a good idea. Often you'll just end up having to redo it anyway when you finally decide to coerce and/or validate arguments to make your code more robust, e,g,

```
def initialize(@name, @age, @location); end
```

Becomes

```
def initialize(name, age, location)
  @name = name.to_s
  @age = age.to_i
  @location = Location.new(location)
end
```

Might as well write it out from the get-go in preparation.

**#13 - 10/27/2012 06:34 AM - ko1 (Koichi Sasada)**

*- Target version changed from 2.0.0 to 2.6*

I changed target to next minor.
I think someone who want to introduce it need to persuade matz.

**#14 - 12/09/2012 01:31 PM - Anonymous**

Well... I like the sweetness... But to have such a feature working syntactically from
inside of #initialize method, but not from other methods... I don't know.

It's not like this is my suggestion, but since it is up for discussion, let me try to
straighten this proposal according to my own thinking:

These "attributes on steroids" are a thing to be done not at the level of the #initialize
method, but at the level of the module, along with #attr_accessor and friends.

Imho, it would be necessary to update #attr_accessor & friends to accept :autoinit
named argument:

```
attr_reader :name, :age, autoinit: true
```

Now there are two flavors of this candy, one with ordered arguments, one with named.
So we could have to specify it:

```
attr_reader :name, autoinit: :ordered
attr_reader :age, autoinit: :named
```

I'm sure you know what I mean here. Default option could be eg. :named.

Also, #autoinit method, or rather, #autoinit_named, #autoinit_ordered,
would have to be added to the Module, for those times, when we want to
autoinit, but don't want the reader/writer/accessor:

```
autoinit_named :age
autoinit_ordered :name
```

Afaik, current #attr_accessor & friends work by defining instance methods on
the module. How #autoinit should work, is a question. Two possibilities come
to my mind:

1. By patching #initialize method.

2. By creating and including a mixin patching #new class method, that would
   set the appropriate instance variables right after creating a new instance,
   in effect something like this:

   ```
   module AgeNamedArgAutoinit
     def new *args, &block
       named_args = args.extract_options!
       age = named_args.delete :age
       modified_args = args + named_args.empty? ? [] : [named_args]
       new_instance = super *modified_args, &block
       new_instance.instance_variable_set :@age, age
     end
   end
   ```

   ```
   module NameOrderedArgAutoinit
     def new *args, &block
       name = args.shift
       new_instance = super *args, &block
       new_instance.instance_variable_set :@name, name
     end
   end
   ```

```
class MyClass
  include AgeNamedArgAutoinit
  include NameOrderedArgAutoinit
end
```

Now MyClass.new( "John Smith", :whatever, age: 35, other_stuff: :whatever ) should
behave in the expected way.

Again, I have not come up with this proposal, I do not give +1 or -1 to it,
I am only trying to iron it to be more consistent, leaving the decision to others.

**#15 - 12/09/2012 01:43 PM - phluid61 (Matthew Kerwin)**

alexeymuranov (Alexey Muranov) wrote:

> +1, why only initialize?

I agree.  Is there a reason not to specify something other than a local variable as the receiver for a method parameter?

For example:

```
# precondition: ???
# postcondition: updates @instance_var and $global_var
def some_thing(local_var, @instance_var, $global_var)
  # ...
end
```

I know it's not safe, in that it makes it quite easy to shoot one's self in the foot, but I don't know that it's necessarily a bad thing unless someone tells me it is.

**#16 - 02/26/2013 07:39 AM - sikachu (Prem Sichanugrist)**

I think this is a good feature, so I'd like to support this (and possibly, provide a patch for this)

Reading from all the comments, I saw that someone has some concern about having this feature on another method definition (not just initialize) as well. I think we should implement it for any type of method definition if that's going to make the code cleaner, but the main focus here is for the initialize method.

As per comment 12, I think it's OK to start up with def initialize(@foo, @bar, @baz) and then refactor it if you need to perform any method on those arguments before store it to an instance variable.

I think adding support for only local and instance variables make sense, and I think we need to make sure that we're not supporting global variable like in comment 15.

Anyway, the benefit I'm seeing here is that we're not cluttering the initializer with all those obvious instance variable assignments. I also think it's a good syntactic sugar and it make sense after I saw all the usage from CoffeeScript. I hope I can help to make this happen in the next minor.

**#17 - 06/23/2013 09:40 AM - nobu (Nobuyoshi Nakada)**

*- Description updated*

**#18 - 06/23/2013 09:42 AM - nobu (Nobuyoshi Nakada)**

*- Category set to syntax*

*- Target version changed from 2.6 to Next Major*

**#19 - 03/13/2014 07:10 PM - TylerRick (Tyler Rick)**

I would *love* to see this feature in Ruby.  Assigning an argument to an instance variable in a constructor is something that we do in *almost every constructor we write*, so I think this should be made as easy and simple as possible, by adding a little syntactic sugar to the language.

I shouldn't have to repeat myself and type out each argument name ***3 times*** in every constructor I write, for something as mundane as this:

```
def initialize(name₁, …)
  @name₂ = name₃
  …
end
```

This constant repetition feels inelegant to me and goes against one of the Ruby community's most fundamental values (Don't Repeat Yourself).

This method could be simplified to simply this:

```
def initialize(@name, …)
end
```

I think CoffeeScript solved this problem quite nicely.  Many constructors in CoffeeScript end up being beautiful, simple one-liners!

```
constructor: (@name) ->
```

And with the rising popularity of CoffeeScript, there are going to be more and more Rubyists not only *wishing* for this but also ***expecting*** this same feature to exist in Ruby as well.  :)

---

Here are a few more "votes" for this feature:

- http://stackoverflow.com/questions/10856191/ruby-automatically-set-instance-variable-as-method-argument
- http://stackoverflow.com/questions/9597249/in-ruby-can-i-automatically-populate-instance-variables-somehow-in-the-initializ/10855962
- http://stackoverflow.com/questions/16072965/why-do-method-arguments-not-work-for-assignment

Plus various attempts at removing the duplication from assign variables in constructors, using only pure Ruby:

- http://redsquirrel.com/cgi-bin/dave/dynamic/def_init.html (def_init :arg1, :arg2)
- http://blog.jayfields.com/2007/04/ruby-assigning-instance-variables-in.html (initializer :arg1, :arg2)
- https://github.com/rubyworks/facets/blob/master/lib/core/facets/kernel/assign.rb (assign(hash))
- https://github.com/sheldonh/magic_options (magic_initialize and magic_options(hash)) (But pure Ruby solutions can only go so far, and none of those solutions really solve the problem nicely enough...)

**#20 - 03/14/2014 12:33 AM - nobu (Nobuyoshi Nakada)**

*- Description updated*

Tyler Rick wrote:

> I think CoffeeScript solved this problem quite nicely.  Many constructors in CoffeeScript end up being beautiful, simple one-liners!
>
> ```
> constructor: (@name) ->
> ```

Sorry, it doesn look beautiful to me.

**#21 - 03/14/2014 02:08 AM - phluid61 (Matthew Kerwin)**

Does anyone have a link to discussions/logs that lead to the decision to remove instance/global variables from block parameters?

**#22 - 01/27/2016 09:44 AM - nobu (Nobuyoshi Nakada)**

*- Has duplicate Feature #12023: Allow ivars to be used as method arguments added*

**#23 - 07/10/2016 12:58 PM - nobu (Nobuyoshi Nakada)**

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

**#24 - 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*

**#25 - 10/03/2018 12:12 AM - mame (Yusuke Endoh)**

*- Related to Feature #15192: Introduce a new "shortcut assigning" syntax to convenient setup instance variables added*