Ruby master - Feature #16095

2 Features: remove (simplify) 'new' keyword and Property Shorthand

08/10/2019 09:16 PM - D1mon (Dim F)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Rejected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

**common use:**

```ruby
class A
  def initialize(arg1, arg2)
    @arg1 = arg1
    @arg2 = arg2
  end
end
```

A.new(1,2)

1) feature: remove 'new' keyword:

A(1,2) # shorter and more comfortable

2) feature: Property Shorthand (like ES6)

```ruby
class A
  def initialize(arg1, arg2)
    # shorter and more comfortable
    @arg1, @arg2
    # or this:
    @arg1
    @arg2
  end
end
```

A(1,2)

So as not to duplicate the code (words). May need to do some other syntax or character (:arg1, ^arg1, %arg1, ...)

can also be applied to other types.

**Hash example:**

```ruby
a = 1
b = 2
h = {a:a, b:b} # common use
h = {a,b} # new syntax or {%a,%b} - any syntax of your choice
```

**Related issues:**

Related to Ruby master - Feature #11105: ES6-like hash literals

Rejected

**History**

#1 - 08/10/2019 10:22 PM - jeremyevans0 (Jeremy Evans)

D1mon (Dim F) wrote:

1) feature: remove 'new' keyword:

A(1,2) # shorter and more comfortable

new is a method, not a keyword. You can already get what you want:

```ruby
module Kernel
  private
end
```

03/18/2022
Adding constructor methods to Kernel for every class would bloat the method namespace. There are also already cases where a Kernel method exists with the same name as a core class but has different behavior (e.g. Kernel#Hash vs. Hash.new), so this change could not be backwards compatible.

2) feature: Property Shorthand (like ES6)

Already requested in #5825 and #15192.

can also be applied to other types.

Hash example:

```ruby
a = 1
b = 2
h = {a:a, b:b} # common use
h = {a, b} # new syntax or {%a,%b} - any syntax of your choice
```

Already requested in #15236.

#2 - 08/10/2019 10:40 PM - shevegen (Robert A. Heiler)

The net-gain from removing .new would be quite minimal. I also think it is less readable too, if .new were removed - but even well aside from this, this seems to be backwards incompatible and may have to require a long transition time if it is possible to add in the first place.

I am also not sure whether matz would want to have this - IMO there are too many drawbacks associated with the change, with the net benefit of writing shorter code being a very minimal advantage, if it is one at all. There is also the issue of functions/methods that could be capitalized such as Integer(); I don't think we should have to consider whether this is a method, or instead Integer.new, so personally I am against omitting ".new"; I think .new is fine, and it is four characters - that can not be that much. (Four in the sense of A(1,2) versus A.new(1,2) - there are four characters more in the latter).

Property Shorthand was discussed before; matz responded back then. I believe what he wrote back then still applies. (I don't remember offhand but I think one comment from matz was to wait and see; I think that was in regards to javascript and how they use their shorthand variant.)

It may be better to add to the discussion at #15192 if you feel strongly about shorthand syntax.

As for other syntax such as:

`:arg1, ^arg1, %arg1, ...`

I think some of them are problematic. First one is symbol - I think people would not expect a symbol to behave as a shortcut syntax in this context. But either way I think it is best to discuss this at #15192. IMO.

#3 - 08/11/2019 12:16 AM - znz (Kazuhiro NISHIYAMA)

- Related to Feature #11105: ES6-like hash literals added

#4 - 08/11/2019 07:24 AM - D1mon (Dim F)

Feature 1: I agree that you can do anything with the help of metaprogramming, and this will have to be registered in each library. Therefore, I propose to do this at the language level (Ruby core).

#5 - 08/11/2019 07:31 AM - D1mon (Dim F)

I correctly understood that this function will be added in version 2.7?

https://github.com/ruby/ruby/pull/2231/commits/7da3cdd9aa132307eff0e4376ad6a3819940f2d

#6 - 08/11/2019 08:00 AM - nobu (Nobuyoshi Nakada)

First, please do not mix requests for irrelevant features in one ticket.
D1mon (Dim F) wrote:

I correctly understood that this function will be added in version 2.7?
https://github.com/ruby/ruby/pull/2231/commits/7da3cdc9aa132307eff0e4376ad6a3819940fc2d

It's a just pull-request.
Everyone can request favorite features, but not all will be merged.

#7 - 08/11/2019 08:30 PM - D1mon (Dim F)
figured out how to do it:

%P[arg1, arg2]

in def initialize:

    def initialize(arg1, arg2)
        %P[arg1, arg2] # equals to: @arg1 = arg1, @arg2 = arg2
    end

in hash:

    a = 1
    b = 2
    h = %p[a, b] # equals to: h = {a: a, b: b}

what do you say ???

#8 - 08/12/2019 12:12 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of Feature #15192: Introduce a new “shortcut assigning” syntax to convenient setup instance variables added

#9 - 08/12/2019 12:12 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of Feature #5825: Sweet instance var assignment in the object initializer added

#10 - 08/12/2019 12:13 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of deleted (Feature #5825: Sweet instance var assignment in the object initializer)

#11 - 08/12/2019 12:13 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of deleted (Feature #15192: Introduce a new “shortcut assigning” syntax to convenient setup instance variables)

#12 - 08/12/2019 12:15 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of Bug #7522: Non-core "Type()" Kernel methods return new objects added

#13 - 08/12/2019 12:16 AM - nobu (Nobuyoshi Nakada)
- Is duplicate of deleted (Bug #7522: Non-core "Type()" Kernel methods return new objects)

#14 - 12/01/2019 07:56 AM - jwmittag (Jörg W Mittag)
Please, do not mix requests for multiple unrelated features in one feature request. It makes it impossible to properly address them. For example, what should happen with this feature request if one feature gets accepted and one gets rejected?

Also, in this particular case, all three of the features you request have been requested before, but it is impossible to properly mark this feature request as a duplicate because which of the three duplicated features should you close it as a duplicate of?

D1mon (Dim F) wrote:

    common use:

    class A
        def initialize(arg1, arg2)
            @arg1 = arg1
            @arg2 = arg2
        end
    end
    A.new(1, 2)

    1) feature: remove 'new' keyword:
There is no such thing as a new keyword in Ruby. It's just a method like any other method. If you want to remove this method, you need to propose a different method as a replacement. For example, like this:

```ruby
class Class
  alias_method :foo, :new
end
A.foo(1, 2)
```
or something like this

```ruby
class Class
  alias_method [], :new
end
A[1, 2]
```
or something like this

```ruby
class Class
  alias_method :call, :new
end
A.(1, 2)
```

It seems you simply want a method with the same name as the constant referencing the class. You can easily do that, too:

```ruby
def A(...)
  A.new(...)
end
A(1, 2)
```

There is no need for a new language feature, you can easily implement what you want using the existing language features.

Note that this would completely break backwards-compatibility because there are already existing methods `Kernel<Array, Kernel<Complex, Kernel<Float, Kernel<Hash, Kernel<Integer, Kernel<Rational>, and Kernel<String in the core library, as well as Kernel<BigDecimal in the standard library and probably others in third-party libraries and gems.

In fact, the idiom "method with the same name as the class" already has a very specific meaning and you are completely breaking that meaning.

2) feature: Property Shorthand (like ES6)

```ruby
class A
  def initialize(arg1, arg2)
    # shorter and more comfortable
    @arg1, @arg2
    # or this:
    @arg1
    @arg2
  end
end
```

This completely breaks backwards-compatibility, since `@arg2` already has an existing meaning: dereference and evaluate @arg2. It would basically make it impossible to write simple getters, e.g.:

```ruby
def visible?
  @is_visible
end
```

Just as a historical data point, I would like to point out that up until Ruby 1.8.7, it was possible to write

```ruby
class A
  define_method(:initialize) do |
    @arg1, @arg2
  end
end
```

and I have never seen it used even once. That is an indicator as to how important the Ruby community sees such a feature: even when it existed, nobody used it.

So as not to duplicate the code (words). May need to do some other syntax or character (\-arg1, ^arg1, %arg1, ...)
At least one, maybe two, of these conflict with existing syntax.

#15 - 12/01/2019 08:40 AM - nobu (Nobuyoshi Nakada)
- Status changed from Open to Rejected

One of the requests is a duplicate of an already rejected request.