Hash value omission

How about to allow value omission in Hash literals:

```ruby
x = 1
y = 2
h = {x:, y:}
p h #=> {:x=>1, :y=>2}
```

And in keyword arguments:

```ruby
def login(username: ENV["USER"], password:)
  p(username:, password:)
end
login(password: "xxx") #=> {:username=>"shugo", :password=>"xxx"}
```

Related issues:
- Related to Ruby master - Feature #11105: ES6-like hash literals
  - Rejected
- Related to Ruby master - Feature #18124: Hash shorthands (matching constructo...
  - Closed
- Related to Ruby master - Feature #14973: Proposal of percent literal to expan...
  - Closed
- Has duplicate Ruby master - Feature #17292: Hash Shorthand / Punning
  - Closed

Associated revisions

Revision 78998668 - 09/11/2021 10:09 AM - nobu (Nobuyoshi Nakada)
Another test for [Feature #14579]

The value of the dynamic key cannot be omitted for now.

Revision d05ef388 - 09/11/2021 10:20 AM - nobu (Nobuyoshi Nakada)
[DOC] NEWS for [Feature #14579] [ci skip]

Revision 297f9b8d - 09/11/2021 11:23 AM - shugo (Shugo Maeda)
Add documentation and tests for keyword argument value omission

[Feature #14579]

History

#1 - 03/06/2018 01:50 PM - shugo (Shugo Maeda)
- Related to Feature #11105: ES6-like hash literals added

#2 - 03/06/2018 01:51 PM - shugo (Shugo Maeda)
- File hash_value_omission.diff added

#3 - 03/08/2018 11:04 AM - Eregon (Benoit Daloze)
I find this syntax very confusing.

The two occurrences of password: above means two very different things (required kwarg and auto-hash creation).

For

```ruby
x = 1
```
It looks to me like the values are missing. I'd prefer a syntax which is different than "key syntax without value", and refers to the variable name used for the value more clearly, like:

```ruby
x = 1
y = 2
h = {x, y}
```

That would also work for the second case like so:

```ruby
def login(username: ENV['USER'], password:)
  p({username, password})
end
```

Should this also work for non-Symbols keys like:

```ruby
x = 1
y = 2
h = { 'x' => , 'y' => }
```

I'd prefer a syntax which is different than "key syntax without value", and refers to the variable name used for the value more clearly, like:

```ruby
x = 1
y = 2
h = {x, y}
```

Please no, this is too close to perl's weird handling of lists/hashes. To me it reads like you're trying to write:

```ruby
h = {1=>2}
```

I understand the suggestion trying to make the syntax even more succinct (less to type) but I think it's one step too much. Ruby already has a quite condensed syntax.

I think this syntax here, asked by Benoit, is also problematic:

```ruby
h = { "x" => , "y" => }
```

Has a slight "visual" problem, at the least to me. I would expect => to "point" to something on the right hand side, which the normal syntax in hashes, in ruby, requires (unless you use the foo: :bar syntax notation).

The:

```ruby
h = {x:, y:}
```

to my brain it's indeed a bit confusing because I would normally expect something on the right side of "foo: ".

I proposed the above syntax in #11105, but it was rejected, and this proposal is alternative.
I prefer this syntax to #11105, but this introduces a Ruby-specific syntax different from ES6 syntax. Besides that, I don't like both anyway because they are not intuitive (for me).

Matz.

-I prefer this syntax to #11105, but this introduces a Ruby-specific syntax different from ES6 syntax. Besides that, I don't like both anyway because they are not intuitive (for me).

So I withdraw this proposal.

Our mindset has been updated (mostly due to mandatory keyword arguments).

Accepted.

Matz.

For the record: { '#{ str }': } is not allowed. Matz said that it is intentional.

We should allow it to call a (private) method if no variable with the name defined. We use methods in RSpec or with attr_reader that look like variables, and programmers don't necessarily distinguish between methods from variables when writing a program. I believe this syntax should take methods into account.
We should allow it to call a (private) method if no variable with the name defined. We use methods in RSpec or with attr_reader that look like variables, and programmers don't necessarily distinguish between methods from variables when writing a program. I believe this syntax should take methods into account.

I would agree that (private) methods are very useful here, especially attr_* methods. There are a few cases I would wonder what they do:

- @var: - Would this work with instance/class/global/constant variables if they're valid symbols?
- a = 1; [a:, b: 3] - Does it support mixing omissions and regular values?
- p a:, b: 3 - Does it work with implied hashes / keywords? (I think yes).

I agree that `{ #{ str }:` should not be allowed, as it presents potential for abuse and vulnerabilities.

I've PR'd the second case on mixed values, but just considered the first with ivars and similar concepts. I'm not sure which way that should go.

We also discussed further with Matz and concluded that quoted keys ({ "key": }) are not allowed with or without interpolation. This is simply because you don't need that when any local variable or constant can be written without quotation, and because it might make you feel it could possibly mean { "key": "key" } and that would be confusing.

@var: - Would this work with instance/class/global/constant variables if they're valid symbols?

No, because we didn't change the symbol key syntax. { @var: @var } is not valid, so { @var: } isn't either. The same goes for $var and @@var.

- a = 1; [a:, b: 3] - Does it support mixing omissions and regular values?

Yes.

- p a:, b: 3 - Does it work with implied hashes / keywords? (I think yes).

Yes, but beware when you omit the last value without the closing parenthesis. The interpreter will look further past the line end for a value.

I assumed the value should be a local variable. The merged patch calls the method when the local variable is not defined. I doubt this is sufficient behavior. Any opinion?

I believe a method should be called when a local variable is not defined. Because it's convenient as knu stated, and because {x:} is a syntax sugar of {x: x} except that keywords are allowed.

Note that constants are also allowed:

```ruby
X = 1
p(X:) #=> {X=>1}
```

except that keywords are allowed.

I meant that keywords are allowed as local variable or method names. For example, {if:} is not a syntax error and {self:} doesn't access the receiver but a local variable or method self.
I meant that keywords are allowed as local variable or method names.
For example, {if:} is not a syntax error and {self:} doesn't access the receiver but a local variable or method self.

Ah, so {if:} means something close to {if: local_variable_get(:if)} and '{self:} means something close to {self: local_variable_get(:self)} (and NOT {self: self}). Not sure we need this, but also not sure it hurts.

Ah, so {if:} means something close to {if: local_variable_get(:if)} and '{self:} means something close to {self: local_variable_get(self)} (and NOT {self: self}).

Yes.
Technically speaking, send{if:} is used instead of local_variable_get if the local variable is not defined.

Not sure we need this, but also not sure it hurts.

In the meeting just after RubyKaigi, someone pointed out that {if:}[:if] is faster than binding.local_variable_get(:if).

I don't think using if as the name of a local variable is a good idea, and I don't think the speed of bad ideas should concern us too much.

@duerst (Martin Dürst)
I don't think using if as the name of a local variable is a good idea,
It is good (and widely used, BTW) name for a method parameter, in contexts like
defect :foo, if: :something?
I don't see how it is bad idea, while producing the clearest method call convention for "conditional" DSLs.

#29 - 09/15/2021 10:08 AM - shugo (Shugo Maeda)

duerst (Martin Dürst) wrote in #note-27:

I don't think using if as the name of a local variable is a good idea, and I don't think the speed of bad ideas should concern us too much.

As zverok stated, a keyword such as if is used as a keyword argument (especially on Rails?).

#30 - 09/16/2021 01:41 AM - Dan0042 (Daniel DeLorme)

matz (Yukihiro Matsumoto) wrote in #note-16:

I assumed the value should be a local variable.

I also assumed the same thing, but after getting over my initial surprise I found this way is quite nice, very ruby-ish. A bit like learning that rescue => obj.attr is valid.

#31 - 09/16/2021 06:10 AM - shugo (Shugo Maeda)

- Status changed from Assigned to Closed

Matz accepted the current behavior at DevelopersMeeting20210916Japan

#32 - 09/16/2021 09:22 AM - knu (Akinori MUSHA)

...Which is that { symbol: } verbosely means { symbol: binding.local_variable_defined?(:symbol) ? binding.local_variable_get(:symbol) : __send__(:symbol) } with no exception, no matter if the symbol is if, self, fork, return or whatever.

#33 - 11/30/2021 05:00 AM - hsbt (Hiroshi SHIBATA)

- Related to Feature #14973: Proposal of percent literal to expand Hash added

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>hash_value_omission.diff</td>
<td>619 Bytes</td>
<td>03/06/2018</td>
<td>shugo (Shugo Maeda)</td>
</tr>
</tbody>
</table>