# Ruby trunk - Feature #14183

## "Real" keyword argument

12/14/2017 06:59 AM - mame (Yusuke Endoh)

| | |
|---|---|
| **Status:** | Open |
| **Priority:** | Normal |
| **Assignee:** | |
| **Target version:** | Next Major |

**Description**

In RubyWorld Conference 2017 and RubyConf 2017, Matz officially said that Ruby 3.0 will have "real" keyword arguments. AFAIK there is no ticket about it, so I'm creating this (based on my understanding).

In Ruby 2, the keyword argument is a normal argument that is a Hash object (whose keys are all symbols) and is passed as the last argument. This design is chosen because of compatibility, but it is fairly complex, and has been a source of many corner cases where the behavior is not intuitive. (Some related tickets: #8040, #8316, #9898, #10856, #11236, #11967, #12104, #12717, #12821, #13336, #13647, #14130)

In Ruby 3, a keyword argument will be completely separated from normal arguments. (Like a block parameter that is also completely separated from normal arguments.)
This change will break compatibility; if you want to pass or accept keyword argument, you always need to use bare sym: val or double-splat ** syntax:

```
# The following calls pass keyword arguments
foo(..., key: val)
foo(..., **hsh)
foo(..., key: val, **hsh)

# The following calls pass **normal** arguments
foo(..., {key: val})
foo(..., hsh)
foo(..., {key: val, **hsh})

# The following method definitions accept keyword argument
def foo(..., key: val)
end
def foo(..., **hsh)
end

# The following method definitions accept **normal** argument
def foo(..., hsh)
end
```

I think here is a transition path:

- Ruby 2.6 (or 2.7?) will output a warning when a normal argument is interpreted as keyword argument, or vice versa.
- Ruby 3.0 will use the new semantics.

**Related issues:**

| | | |
|---|---|---|
| Related to Backport200 - Backport #8040: Unexpect behavior when using keyword... | **Closed** | **03/08/2013** |
| Related to Ruby trunk - Bug #8316: Can't pass hash to first positional argume... | **Closed** | |
| Related to Ruby trunk - Bug #9898: Keyword argument oddities | **Closed** | **06/03/2014** |
| Related to Ruby trunk - Bug #10856: Splat with empty keyword args gives unexp... | **Closed** | |
| Related to Ruby trunk - Bug #11236: inconsistent behavior using ** vs hash as... | **Open** | |
| Related to Ruby trunk - Bug #11967: Mixing kwargs with optional parameters ch... | **Rejected** | |
| Related to Ruby trunk - Bug #12104: Procs keyword arguments affect value of p... | **Rejected** | |
| Related to Ruby trunk - Bug #12717: Optional argument treated as kwarg | **Open** | |
| Related to Ruby trunk - Bug #12821: Object converted to Hash unexpectedly und... | **Closed** | |

| | |
|---|---|
| Related to Ruby trunk - Bug #13336: Default Parameters don't work | **Open** |
| Related to Ruby trunk - Bug #13647: Some weird behaviour with keyword arguments | **Feedback** |
| Related to Ruby trunk - Bug #14130: Keyword arguments are ripped from the mid... | **Open** |

### History

**#1 - 12/14/2017 07:23 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Backport #8040: Unexpect behavior when using keyword arguments added*

**#2 - 12/14/2017 07:23 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #8316: Can't pass hash to first positional argument; hash interpreted as keyword arguments added*

**#3 - 12/14/2017 07:23 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #9898: Keyword argument oddities added*

**#4 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #10856: Splat with empty keyword args gives unexpected results added*

**#5 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #11236: inconsistent behavior using ** vs hash as method parameter added*

**#6 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #11967: Mixing kwargs with optional parameters changes way method parameters are parsed added*

**#7 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #12104: Procs keyword arguments affect value of previous argument added*

**#8 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #12717: Optional argument treated as kwarg added*

**#9 - 12/14/2017 07:24 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #12821: Object converted to Hash unexpectedly under certain method call added*

**#10 - 12/14/2017 07:25 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #13336: Default Parameters don't work added*

**#11 - 12/14/2017 07:25 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #13647: Some weird behaviour with keyword arguments added*

**#12 - 12/14/2017 07:25 AM - hsbt (Hiroshi SHIBATA)**

*- Related to Bug #14130: Keyword arguments are ripped from the middle of hash if argument have default value added*

**#13 - 12/14/2017 08:27 AM - jeremyevans0 (Jeremy Evans)**

For a method definition like:

```
def foo(hsh={})
end
```

Will either of the following continue to work?:

```
foo(key: val)
foo(:key => val)
```

One performance issue with keyword arguments is that keyword splats allocate a hash per splat, even if no keywords are used.

In performance sensitive code, allocations can be avoided using a shared frozen hash as the default argument:

```
OPTS = {}.freeze
def foo(hsh=OPTS)
  bar(1, hsh)
end
def bar(val, hsh=OPTS)
end
```

By doing this, calling foo without keyword arguments does not allocate any hashes even if the hash is passed to other methods.  If you use keyword arguments, you have to do:

```
def foo(**hsh)
  bar(1, **hsh)
end
def bar(val, **hsh)
end
```

Which I believe allocates a multiple new hashes per method call, one in the caller and one in the callee.  Example:

```
require 'objspace'
GC.start
GC.disable
OPTS = {}

def hashes
  start = ObjectSpace.count_objects[:T_HASH]
  yield
  ObjectSpace.count_objects[:T_HASH] - start - 1
end

def foo(opts=OPTS)
  bar(opts)
end
def bar(opts=OPTS)
  baz(opts)
end
def baz(opts=OPTS)
end

def koo(**opts)
  kar(**opts)
end
def kar(**opts)
  kaz(**opts)
end
def kaz(**opts)
end

p hashes{foo}
p hashes{foo(OPTS)}
p hashes{koo}
p hashes{koo(**OPTS)}

# Output
0
0
5
6
```

I humbly request that unless keyword splats can be made to avoid allocation, then at least make:

```
def foo(hsh)
end
foo(:key => val)
```

still function as it has since ruby 1.8, since that can be considered a hash and not a keyword argument.

#### #14 - 12/18/2017 01:42 PM - mame (Yusuke Endoh)

*- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN)*

*- Tracker changed from Bug to Feature*

#### #15 - 01/16/2018 05:33 PM - sos4nt (Stefan Schüßler)

I've filed a bug report some time ago, maybe you could add it as a related issue: https://bugs.ruby-lang.org/issues/11993

#### #16 - 01/19/2018 03:10 AM - dsferreira (Daniel Ferreira)

It's not clear for me all the implications of this change.
Would it be possible to exemplify the before and after behaviours in the description?

It feels to me that with this implementation it would be possible to consider both symbols and strings as keys for the keywords hash.

Would it be a possibility?

The dynamic generation of keywords hashes would be positively impacted with that move.