Ruby master - Bug #11993

foo(hash) is handled like foo(**hash)

01/15/2016 09:05 AM - sos4nt (Stefan Schüßler)

Status: Rejected
Priority: Normal
Assignee:
Target version:
ruby -v: 2.3.0
Backport: 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

Given this method:

```ruby
def foo(a = nil, b: nil)
  p a: a, b: b
end
```

I can pass keyword arguments to it and I can turn a hash into keyword arguments with the `**` operator:

```ruby
foo(b: 1)       #=> {:a=>nil, :b=>1}
foo(**{b: 1})   #=> {:a=>nil, :b=>1}
```

What baffles me is that a hash is also turned into keyword arguments without the `**` operator:

```ruby
foo({b: 1})     #=> {:a=>nil, :b=>1}
```

This looks like a flaw to me. I was expecting:

```ruby
foo({b: 1})     #=> {:a=>{:b=>1}, :b=>nil}
```

Which would have resembled the way `*` works:

```ruby
def bar(a = nil, b = nil)
  p a, b
end
```

```ruby
bar(1, 2)       #=> {:a=>1, :b=>2}
bar(['1', '2']) #=> {:a=>1, :b=>2}
bar([1, 2])     #=> {:a=>['1', '2'], :b=>nil}
```

But currently, there doesn't seem to be a difference between `foo(hash)` and `foo(**hash)`.

Is this behavior intended? If so, what's the rationale behind this decision?

History

#1 - 01/31/2016 05:43 AM - avit (Andrew Vit)
See #11967 for Marc-Andre's explanation.

#2 - 09/05/2018 04:45 PM - marcandre (Marc-Andre Lafortune)
- Status changed from Open to Rejected

First, `foo(b: 1)` has been exactly the same as `foo({b: 1})` since Ruby 1.8 at least. It is parsed exactly the same way. It's syntax sugar.

```ruby
require 'ripper'
Ripper.sexp('foo(a : 1)') == Ripper.sexp('foo({a : 1})')  # => true
```

As you note, the `**` operator is not needed in many cases, but there are cases where it matters. You can see a difference in these three cases:

a) It merges keyword arguments:

```ruby
h = {a: 1, b: 2}
```
\[ p(h, c: 3) \Rightarrow \{ a: 1, b: 2\}, \text{then } \{ c: 3\} \]
\[ p(**h, c: 3) \Rightarrow \{ a: 1, b: 2, c: 3\} \]

b) It insures that a hash is viewed as keyword arguments:

\[
\begin{align*}
h &= \{ 'a' => 1 \} \\
p(h) &\Rightarrow \{ 'a' => 1 \} \\
p(**h) &\Rightarrow \text{TypeError (hash key "a" is not a Symbol)}
\end{align*}
\]

c) It differentiates between an actual empty hash {} and nothing at all:

\[
\begin{align*}
def \text{foo}(x) \\
  p x
end
\end{align*}
\]
\[
\begin{align*}
e &= \{} \\
\text{foo('hello', **e) } &= 'hello' \\
\text{foo('hello', e) } &\Rightarrow \text{ArgumentError (wrong number of arguments (given 2, expected 1))}
\end{align*}
\]

Note that some corner cases may not perfectly handled yet (#15078)

In summary: using ** improves legibility by making the intention crystal clear, makes your code stricter and allows you to easily merge options. There is also discussion to make the use of ** required in some cases in Ruby 3.0 (see #14183).

I'm closing this, but will reopen if need be.