Ruby master - Bug #11993

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

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

<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>2.3.0</td>
</tr>
<tr>
<td>Backport:</td>
<td>2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN</td>
</tr>
</tbody>
</table>

Description
Given this method:

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:

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:

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

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

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

Which would have resembled the way * works:

def bar(a = nil, b = nil)
  p a, b: b
end

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 Lafontune)
- 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.

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:

h = {a: 1, b: 2}
b) It insures that a hash is viewed as keyword arguments:

```ruby
h = {'a' => 1}
p(h) # => {"a"=>1}
p(**h) # => TypeError (hash key "a" is not a Symbol)
```

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

```ruby
def foo(x)
  p x
end

e = {}
foo('hello', **e) # => 'hello'
foo('hello', e) # => ArgumentError (wrong number of arguments (given 2, expected 1))
```

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.