Description

We have Hash#transform_keys and its bang version to change the keys of a hash, but that requires passing a block, which assumes that the mapping from the old keys to the new keys follows some rule. But in reality, we frequently want to change the keys where it is difficult to provide a rule. For example, suppose we have:

```ruby
```

and want to achieve:

```ruby
```

I request an option to change the keys of a hash not by giving a block, but by passing a hash. I came up with two options.

1. Argument for Hash#transform_keys and its bang version

Allow Hash#transform_keys to optionally take a hash argument instead of a block.

```ruby
hash.transform_keys({created: :created_at, updated: :update_time})
```

2. Argument for Hash#slice and the counterparts in other classes

Since Hash#slice is often the first step of modifying a hash into some other hash form, it makes sense to let it take an optional hash argument.

```ruby
hash.slice(:created, :author, transform_keys: {created: :created_at})
# => {created_at: 2019-10-23 17:54:46 +0900, author: "foo"}
```

With option 1, it could make sense to even allow a hash argument and a block simultaneously:

```ruby
hash.transform_keys({created: :created_at, updated: :update_time}, &:to_s)
# => {"created_at" => 2019-10-23 17:54:46 +0900, "update_time" => 2019-10-23 17:59:18 +0900, "author" => "foo"}
```

One quick question: what should happen if both a block and an argument are passed at once?
shyouhei (Shyouhei Urabe) wrote:

Understand the motivation (maybe that of #slice can be separated into another request).

One quick question: what should happen if both a block and an argument are passed at once?

Actually, I just came up with an additional idea regarding that exact point. Updated the issue.

#6 - 10/23/2019 09:26 AM - sawa (Tsuyoshi Sawada)
- Description updated

String#gsub also can take a block or a hash. Using a hash for String#gsub isn't possible in Perl or Python, but can be extremely handy. Maybe the behavior when there's both a block and a hash could be the same as for String#gsub?

#8 - 10/23/2019 07:46 PM - shevegen (Robert A. Heiler)

Personally I have had a need to transform keys (and values) in a hash quite a bit. We also have strange thingies such as HashWithIndifferentAccess so that may be indicative of people wondering about strings/symbols as keys for a hash in general. :) 

To sawa's suggestion: if the choice is solely between option 1 and option 2, I personally would favour 1, mostly because I only rarely use slice in general (primarily I may use slice in regards to String, but even then I tend to prefer e. g. [] start, end positions normally, even if it may not be the same, such as if we include .slice!() use cases). But this may differ on an individual choice, by different ruby users, so I can not generalize it; it is only an opinion if the choice is purely between #1 and #2 alone.

I do not know how frequent the use case may be, but ignoring this for the moment I think this suggestion may be useful.

#9 - 12/20/2019 07:28 AM - knu (Akinori MUSHA)

Which does hash1.transform_keys(hash2) iterate over, hash1's keys or hash2's keys?

#10 - 12/20/2019 07:31 AM - matz (Yukihiro Matsumoto)

I understand. Accepted.
FYI, we do not accept a similar change to transform_values.

Matz.

#11 - 12/20/2019 02:54 PM - nobu (Nobuyoshi Nakada)
- Target version set to 2.8

#12 - 12/23/2019 04:53 AM - duerst (Martin Dürst)

matz (Yukihiro Matsumoto) wrote:

FYI, we do not accept a similar change to transform_values.

Matz - Do you mean "we reject a similar change to transform_values", or do you mean "we have not yet accepted a similar change to transform_values (but could consider it later)?"

#13 - 12/23/2019 08:54 AM - sawa (Tsuyoshi Sawada)

As for the behavior when both a hash argument and a block are given, I suggested in the issue to have the block applied after the hash has applied:

Original proposal

```ruby
hash.transform_keys({created: :created_at, updated: :update_time}, &to_s)
# => {"created_at" => ..., "update_time" => ..., "author" => ...}
```

But I reconsidered this, and now I think that the block should be applied to the residue of what the hash has applied. (In other words, the hash and the
block should be applied mutually exclusively of each other, with the hash having priority over the block.) Hence, I expect the following results:

New proposal

```ruby
hash.transform_keys({created: :created_at, updated: :update_time}, &:to_s)
# => {created_at => ..., update_time => ..., "author" => ...}
```

```ruby
hash.transform_keys({created: "created_at", updated: "update_time"}, &:to_s)
# => {"created_at" => ..., "update_time" => ..., "author" => ...}
```

The reason is twofold.

First, my original proposal would lead to redundancy; I would have to provide the intermediate key :created_at, :update_time, knowing that they will not appear in the final output because of the further transformation of them into strings due to the block. Providing the final keys "created_at" and "update_time" from the beginning would be more straightforward, and will save some internal calculations to be done by Ruby.

Second, the new proposal will have more expressive power. Suppose I actually wanted:

```ruby
{:created_at => ..., :update_time => ..., "author" => ...}
```

That can be done with the new proposal, but not with my original proposal.

---

#14 - 12/23/2019 08:57 AM - sawa (Tsuyoshi Sawada)

knu (Akinori MUSHA) wrote:

Which does hash1.transform_keys(hash2) iterate over, hash1’s keys or hash2’s keys?

My intention of including the key :author in hash was to show that the iteration is done over the receiver (your hash1), not the hash argument (your hash2).

---

#15 - 01/14/2020 12:24 PM - znz (Kazuhiro NISHIYAMA)

- Status changed from Open to Closed