Ruby master - Feature #15492

Let #dig take a "default value" block like Hash#fetch does

01/01/2019 11:14 PM - TylerRick (Tyler Rick)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

fetch provides multiple ways to handle the case where a key can't be found:

> If the key can't be found, there are several options: With no other arguments, it will raise a KeyError exception; if default is given, then that will be returned; if the optional code block is specified, then that will be run and its result returned.

```
{a: 'a'}.fetch(:d) { 'default' }
#=> "default"

{a: 'a'}.fetch(:d) {|key| key }
#=> :d
```

dig obviously can't let you specify a default value as a positional argument like fetch does, but couldn't it at least let you specify a default value by passing a block?

The fact that it currently just silently *ignores* the block that you pass to dig could be misleading, as one might assume (as I did at first) that it's going to return that in case any of the key lookups fail.

Current/desired behavior:

```
{a: {b: {c: 'c'}}}.dig(:a, :b, :d) {|key| key }
#=> nil
# wish it returned :d

{a: {b: {c: 'c'}}}.dig(:a, :b, :d) {|key| 'default' }
#=> nil
# wish it returned 'default'
```

There isn't currently a nice way to do this (that I can think of). I guess you could mix dig and fetch (or chain a bunch of fetchs), but that loses the simple elegance of dig:

```
object.dig(:a, :b)&.fetch(:d) { 'default' }
#=> "default"
```

Note, of course, that if we added default-block behavior to dig, its behavior would be slightly different: it would return the result of the default block if any intermediate step were nil, not just if the last lookup were nil.

```
object = {a: {b: {c: 'c'}}}
object.dig(:x, :y, :z) { 'default' }
#=> "default"
```

Example use case

Sometimes you start out with a simple Hash but over time you may end up moving keys into sub-hashes.

You might have started out with something like

```
config.fetch(:something) { 'default' }
```

But after you move :something under :settings, you have to use dig instead of fetch:

```
config.dig(:settings, :something)
```

The problem is, how do you keep the default value now that it's in a sub-hash? Can't just use || 'default' because the stored value might be false.

Currently I'm working around this with an explicit nil? check...

```
value = config.dig(:settings, :something)
value.nil? ? 'default' : value
```

(Even *that* may not be good enough if one wanted to distinguish between a nil value stored in the object and the case where the key can't be found (a "cache miss"). Having a default block would let you distinguish between a missing key and a nil value stored at the key, in case that distinction were important...)

History

#1 - 01/01/2019 11:17 PM - TylerRick (Tyler Rick)

- Description updated