Ruby - Feature #12116

`Fixnum#divmod`, `Bignum#divmod` with multiple arguments

02/26/2016 06:04 PM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Sometimes, I need to apply divmod repeatedly. For example, in order to convert a number expressing seconds into approx year, day, hour, minutes, seconds (approx in the sense of ignoring leap day and leap second), I can repeatedly apply divmod:

```
seconds = 289342751
minutes, seconds = seconds.divmod(60) # => [4822379, 11]
hours, minutes = minutes.divmod(60) # => [80372, 59]
days, hours = hours.divmod(24) # => [3348, 20]
years, days = days.divmod(365) # => [9, 63]
```

so that I get that 289342751 seconds is approx 9 years 63 days 20 hours 59 minutes and 11 seconds. But it is cumbersome to do all that. It would be convenient if divmod can take multiple arguments so that the conventional divmod is applied from the right-most argument to the left, returning the above result at once:

```
289342751.divmod(365, 24, 60, 60) # => [9, 63, 20, 59, 11]
```

In general, when n arguments are passed to the proposed divmod, an array of n + 1 elements should be returned.

Another use case is nested arrays. Some people tend to express a matrix as a nested array, and try to access the innermost elements using multiple indices. To list the coordinates of the occurrences of 1, one may do:

```
m = [
  [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1],
]

a = []
m.each_with_index do |row, i|
  row.each_with_index do |e, j|
    a.push([i, j]) if e == 1
  end
end
a # => [[0, 0], [1, 1], [2, 2]]
```

But it is often easier to have a flat array and use divmod with it:

```
m = [
  1, 0, 0,
  0, 1, 0,
  0, 0, 1,
]

m.each.with_index.select{|e, _| e == 1}.map{|_, i| i.divmod(3)} # => [[0, 0], [1, 1], [2, 2]]
```

However, once the nesting achieves another level, it becomes cumbersome. Instead of using a nested array:

```
t = [
  [
    ["a", "b"],
    ["c", "d"],
  ],
  [
    ["a", "b"],
    ["c", "d"],
```

```
  ],
]
```

one can keep using a flat array, but that would require repeated application of divmod to covert between the flat index and the nested index. The proposed feature would also help in such case.

Related issues:

Related to Ruby - Feature #4787: Integer#each_modulo(n)	**Closed**
Related to Ruby - Feature #12447: Integer#digits for extracting digits of pla...	**Closed**

History

#1 - 03/17/2016 09:06 AM - shyouhei (Shyouhei Urabe)

Huge +1 to this. The use case is obvious.

#2 - 03/17/2016 09:54 AM - sawa (Tsuyoshi Sawada)

Shyouhei Urabe wrote:

> Huge +1 to this. The use case is obvious.

Thank you.

Still another use case is converting UNIX file permission from octal form to binary form.

```
r, w, x = 5.divmod(2, 2) # => [1, 0, 1]
r, w, x = 5.divmod(2, 2).map(&:positive?) # => [true, false, true]
```

#3 - 03/17/2016 10:33 AM - Eregon (Benoit Daloze)

Nice idea! +1

#4 - 03/17/2016 06:21 PM - marcandre (Marc-Andre Lafortune)

Very nice!

#5 - 03/21/2016 02:00 PM - sawa (Tsuyoshi Sawada)

My proposal remains as I proposed above, but I just wanted to note that there is another natural way to generalize the notion of divmod with multiple divisors, and that people be aware of the difference.

- The proposed generalization: The list of divisors are applied from **the right to the left** on the **quotient** of the previous application of the conventional divmod. Examples are already given.
- The other generalization of the notion: The list of divisors are applied from **the left to the right** on the **remainder** of the previous application of the conventional divmod.

An example of the second notion is a vending machine program. Suppose the change to give back is 937 yen. There are 500-, 100-, 50-, 10-, 5-, and 1-yen coins. The change is calculated as follows:

```
937.divmod(500) # => [1, 437]
437.divmod(100) # => [4, 37]
37.divmod(50) # => [0, 37]
37.divmod(10) # => [3, 7]
7.divmod(5) # => [1, 2]
```

This gives us [1, 4, 0, 3, 1, 2], which represents that one 500-yen, four 100-yen, zero 50-yen, three 10-yen, one 5-yen, and two 1-yen coins should be payed back to give the change.

This second notion may have other use cases such as converting numerals to roman numerals.

I still feel that the first notion, which I proposed, will find more use cases than the second one, so I retain my proposal as is, but was wondering if someone might think differently.

#6 - 05/31/2016 12:32 PM - mrkn (Kenta Murata)

- Related to Feature #4787: Integer#each_modulo(n) added

#7 - 05/31/2016 02:37 PM - mrkn (Kenta Murata)

- Related to Feature #12447: Integer#digits for extracting digits of place-value notation in any base added