Ruby 初級者向けレッスン第 28 回

okkez @ Ruby 関西

2009 年 06 月 20 日

今回の内容

- 配列
- ハッシュ
- ループ
- 再帰

今回のゴール

- 配列とハッシュの違いを知る
- ループと再帰の違いを知る

配列とは

- Array
- 何でも詰め込める入れ物
- インデックスでアクセス可能
- インデックスは zero origin

例:

```
01: #! /usr/bin/ruby
02: # -*- coding: utf-8 -*-
03:
04: arr = [0, 1, 2, 3, 4, 5]         # => [0, 1, 2, 3, 4, 5]
05: arr = ["a", "b", "c", "d", "e"] # => ["a", "b", "c", "d", "e"]
06: arr = %w[a b c d e]              # => ["a", "b", "c", "d", "e"]
07: # 式展開・バックスラッシュ記法が使用可能
08: arr = %W[a b c d e]              # => ["a", "b", "c", "d", "e"]
09: arr = [:a, :b, :c, :d, :e]       # => [:a, :b, :c, :d, :e]
10: arr = ('a'..'e').to_a            # => ["a", "b", "c", "d", "e"]
```

配列でできること

- 要素の参照
- 要素の追加・削除
- Enumerable モジュールのメソッドを使用できる

例:

```
01: #! /usr/bin/ruby
02:
03: arr = %w[a b c d e] # => ["a", "b", "c", "d", "e"]
04: arr[0]   # => "a"
05: arr[1]   # => "b"
06: arr[2]   # => "c"
07: arr[3]   # => "d"
08: arr[4]   # => "e"
09: arr[5]   # => nil
10: arr[100] # => nil
11:
12: arr[5] = "f"
13: arr      # => ["a", "b", "c", "d", "e", "f"]
14: arr << "g"
15: arr      # => ["a", "b", "c", "d", "e", "f", "g"]
16: arr.push "h", "i", "j"
17: arr      # => ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
18: arr.unshift "Z"
19: arr      # => ["Z", "a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
20:
21: arr.shift # => "Z"
22: arr       # => ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
23: arr.pop   # => "j"
24: arr       # => ["a", "b", "c", "d", "e", "f", "g", "h", "i"]
25: 9.times{ arr.pop }
26: arr       # => []
27:
```

配列でできないこと

- 疎な配列を効率よくに作ること
  - ハッシュを使うとうまくいく場合が多い

例:

```
01: #! /usr/bin/ruby
02:
03: arr = []
04: arr[0]  = 'a'
05: arr[10] = 'b'
06: arr[20] = 'c'
07:
08: arr # => ["a", nil, nil, nil, nil, nil, nil, nil, nil, nil,
09:     #     "b", nil, nil, nil, nil, nil, nil, nil, nil, nil,
10:     #     "c"]
```

ハッシュとは

- key と value
- key は一意でなければならない
- key や value には何でも入れられる
- 特定の key にアクセスするのにかかる時間は O(1)

例:

```
01: #! /usr/bin/ruby
02: # -*- coding: utf-8 -*-
03:
04: h = { 0 => 1, 2 => 3, 4 => 5}
05: h = { "a" => 1, "b" => 2, "c" => 3 }
06: h = {
07:   :a => 'foo',
08:   :b => 'bar',
09:   :c => {
10:     :foo => "foo",
11:     :bar => "bar",
12:   },
13: }
14:
15: h # => {:b=>"bar", :c=>{:foo=>"foo", :bar=>"bar"}, :a=>"foo"}
16:
17: # 1.9 では key がシンボルの場合は以下のように書ける
18: h = { a:'A', b:'B', c:'C' }
19: h = { :a => "A", :b => "B", :c => "C" }
```

ハッシュでできること

- 要素を key で参照
- 要素の追加・削除
- Enumerable モジュールのメソッドを使用できる

例:

```
01: #! /usr/bin/ruby
02:
03: h = {
04:   :a => "AAA",
05:   :b => "BBB",
06:   :c => "CCC",
07: }
08:
09: h             # => {:b=>"BBB", :c=>"CCC", :a=>"AAA"}
10: h[:a]         # => "AAA"
11: h[:d] = "DDD"
12: h             # => {:d=>"DDD", :b=>"BBB", :c=>"CCC", :a=>"AAA"}
13: h.delete(:b) # => "BBB"
14: h             # => {:d=>"DDD", :c=>"CCC", :a=>"AAA"}
```

ループとは

あるコードブロックを何度も実行する方法。Ruby にはいくつかループを実現する構文があるがあまり使われない。

- while / until
- for n in x
- loop
  - これだけメソッド

例:

```
01: #! /usr/bin/ruby
02:
03: i = 0
04: while i < 10
05:   puts i
06:   i += 1
07: end
```

```
08:
09: i = 0
10: until i >= 10
11:   puts i
12:   i += 1
13: end
14:
15: i = 0
16: loop do
17:   puts i
18:   i += 1
19:   break if i >= 10
20: end
```

再帰とは

ある関数の中でその関数自身を再び呼び出すこと
例:

```
01: #! /usr/bin/ruby
02:
03: def factorial(n)
04:   return 1 if n <= 1
05:   n * factorial(n - 1)
06: end
07:
08: puts factorial(5)
09: puts (1..5).inject(&:*)
```

ループと再帰の違い

- ループはスタックを消費しない
- 再帰はスタックを消費する
  - StackOverflow する可能性がある

each

- Array の全要素を順番に処理する
- Hash の全要素を一つずつ (key, value のペア) 処理する
  - 1.8 では処理順は保証されない

- 1.9 ではハッシュに追加した順に処理される
  - ただしこの動作は処理系依存である

例:

```
01: #! /usr/bin/ruby
02:
03: a = [1, 2, 3, 4, 5, 6, 7, 8, 8, 10]
04: a.each do |v|
05:   p v
06: end
07:
08: h = {
09:   :key1 => 1,
10:   :key2 => 2,
11:   :key3 => 3,
12: }
13: h.each do |k, v|
14:   p [k, v]
15: end
```

演習

演習 1

- 疎な配列と同じデータを表現するハッシュを比較する

  # 以下のようにアクセスできるデータを作成して比較する a[0] # => 0 a[10] # => 10 a[90] # => 90

- Ruby であまり for 式が使われない理由を三つ考えてください。
- 階乗を求めるプログラムのループ版を書いてください。

演習 2

フィボナッチ数を計算するプログラムをループ版と再帰版の両方とも書いてください。
例:

```
1 1 2 3 5 8 13 21 ...
```

また、以下の数列の漸化式を求めフィボナッチ数と同じように各項の値を計算するプログラムをループ版と再帰版の両方とも書いてください。
例:

```
1 1 1 3 5 9 17 31 57 ...
```

演習 3

バイナリサーチを再帰で実装してください。以下のいずれかの方法で実装してください。バイナリサーチ自体の説明は Wikipedia などを参照してください。

例:

```
bsearch(arr, target) # => target のインデックスが返ってくる
# or
arr.bsearch(target)  # => target のインデックスが返ってくる
```

まとめ

- 配列とハッシュは使う目的が違います
- ループと再帰も使う目的が違います
- それぞれ適材適所に使いましょう

ここから先へ

- 他の collection クラスについて調べてみる
- Enumerable module のリファレンスを通読する
- リファレンスに載っているコードを写経してみる
- 自分でスクリプトを書くときに使ってみる

今後の情報源

公式 Web サイト
http://www.ruby-lang.org/

Ruby リファレンスマニュアル刷新計画
http://doc.loveruby.net/

日本 Ruby の会
http://jp.rubyist.net/

Rubyist Magazine
http://jp.rubyist.net/magazine/

okkez weblog
http://typo.okkez.net/

Ruby Reference Manual (beta)
http://doc.okkez.net/