

テキスト処理 第3回 (2007-05-08)

田中哲

産業技術総合研究所

情報技術研究部門

`akr@isc.senshu-u.ac.jp`

`http://staff.aist.go.jp/tanaka-akira/textprocess-2007/`

今日の内容

- 前回説明を忘れたことを説明する
- テキストから正規表現にマッチした行を取り出す
ツールを改善する
- それに役に立つRuby特有の要素を学ぶ

だいたい任意の 1文字にマッチする 正規表現 /./

- . (ドット)は改行 (¥n) 以外の任意の 1文字にマッチする
- 例

./ = ~ "a"	0
./ = ~ "z"	0
./ = ~ "¥n"	nil
./ = ~ ""	nil

前回のegrep.rb

```
pattern = ARGV[0]      # 第1引数の取り出し
filename = ARGV[1]     # 第2引数の取り出し
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                                # 正規表現オブジェクトに変換
f = open(filename) # 第1引数のファイルをオープン
while line = f.gets # ファイルの終わりまで一行ずつ読む
  if regexp =~ line # 読み込んだ行は正規表現にマッチするか?
    print line      # マッチしてたら表示
  end
end
f.close            # ファイルをクローズ
```

今回のegrep.rb

- Rubyの機能を使うともっと短く書ける

```
pattern = ARGV.shift
regexp = Regexp.compile(pattern)
ARGF.each {|line|
  print line if regexp =~ line
}
```

今回の `egrep.rb` で使用する機能

- `Array#shift`
- 後置 `if`
- ブロック
- `ARGF`

Array#shift

- 配列の先頭を破壊的に取り出すメソッド

```
a = [1,2,3]
```

```
p a          #=> [1,2,3]
```

```
p a.shift   #=> 1
```

```
p a          #=> [2,3]   a の内容が変わっている
```

- 配列のことを Ruby では Array という
- Array に使えるメソッド shift を Array#shift と書く

egrep.rb で Array#shift を使う

```
pattern = ARGV.shift # 第1引数の取り出し
filename = ARGV[0] # 第2引数の取り出し
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                                     # 正規表現オブジェクトに変換
f = open(filename) # 第1引数のファイルをオープン
while line = f.gets # ファイルの終わりまで一行ずつ読む
  if regexp =~ line # 読み込んだ行は正規表現にマッチするか?
    print line # マッチしたら表示
  end
end
f.close # ファイルをクローズ
```


後置 if

- 英語っぽく「文 if 条件」と書ける
print "x exists¥n" if /x/ =~ "text process"
- 以下とほぼ同じ
if 条件
 文
end
if /x/ =~ "text process"
 print "x exists¥n"
end
- 利点
 - 短く記述できる
 - 英語に慣れている人には読みやすいかも
- 注意: 一行で書く

egrep.rb で後置if を使う

```
pattern = ARGV.shift      # 第1引数の取り出し
filename = ARGV[0]       # 第2引数の取り出し
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                             # 正規表現オブジェクトに変換
f = open(filename)      # 第1引数のファイルをオープン
while line = f.gets     # ファイルの終わりまで一行ずつ読む
  print line if regexp =~ line # 行がマッチしたら表示
end
f.close                 # ファイルをクローズ
```

ブロック

- Ruby の大きな特徴のひとつ
- メソッド引数に加えてブロックを渡せる

```
obj.meth(arg) {  
  ...  
}
```

```
func {  
  ...  
}
```

- ブロック内のコードは実行されずに渡される
- 渡されたメソッドはブロックを呼び出せる
何回呼び出しても、呼び出さなくても良い
- Lisp のクロージャに類似。Cでいえば関数ポインタ

ブロックの例

- egrep.rb で使うもの

```
ARGF.each {|line|  
  print line if regexp =~ line  
}
```

- 無限ループ

```
loop {  
  puts "hello"  
}
```

ブロックの用途

- 制御構造の定義
 - 無限ループ loop
 - 配列の各要素の繰り返す Array#each
 - 1行ずつ読み込むループ IO#each, ARGF.each
 - 配列内を探索して見付からなかったときの処理を指定する Array#index
- 高階関数の定義
 - 配列の各要素を変換する Array#map
 - 条件にあった要素だけを集める Array#find_all
 - ソートの比較関数を渡す Array#sort
- 他にもさまざまな用途がある

無限ループ

- 標準で loop メソッドがある

```
loop {  
  puts "hello"  
}
```

- while とかと違って言語に作り付けではない
- このようなものをユーザも定義できる

無限ループなメソッドを定義

- loop2 を定義する

```
def loop2
  while true
    yield # 与えられたブロックを呼び出す
  end
end
```

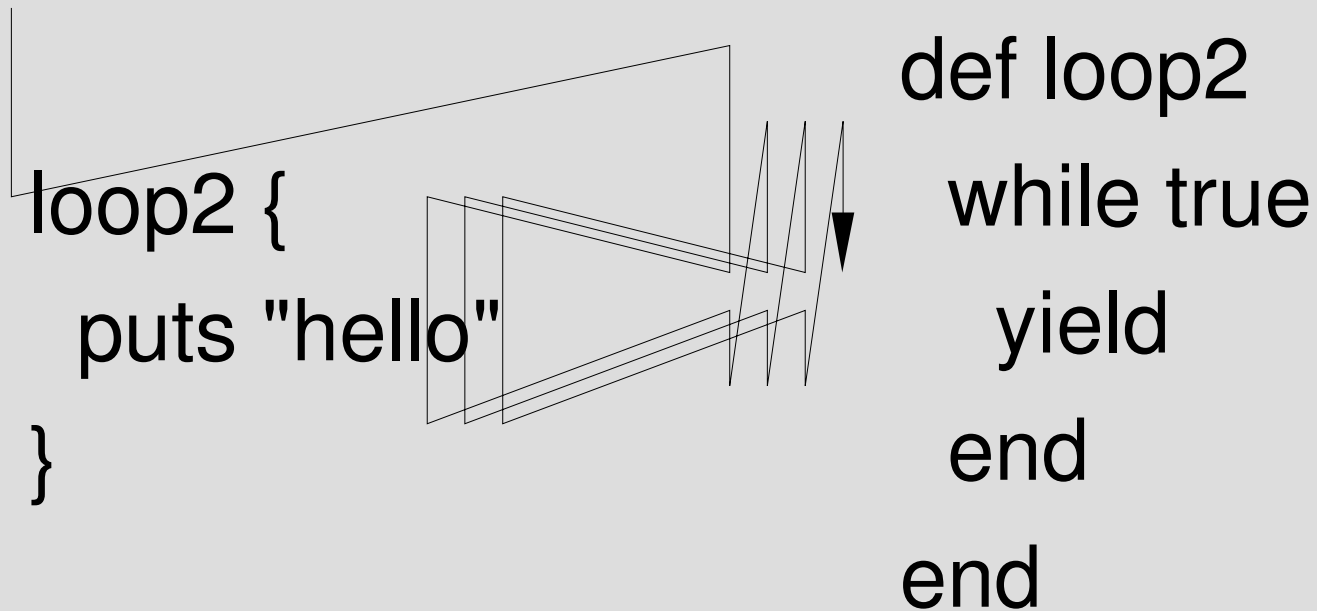
yield

- yield はメソッドに与えられたブロックを呼び出す

```
loop2 {  
  puts "hello"  
}  
  
def loop2  
  while true  
    yield  
  end  
end
```

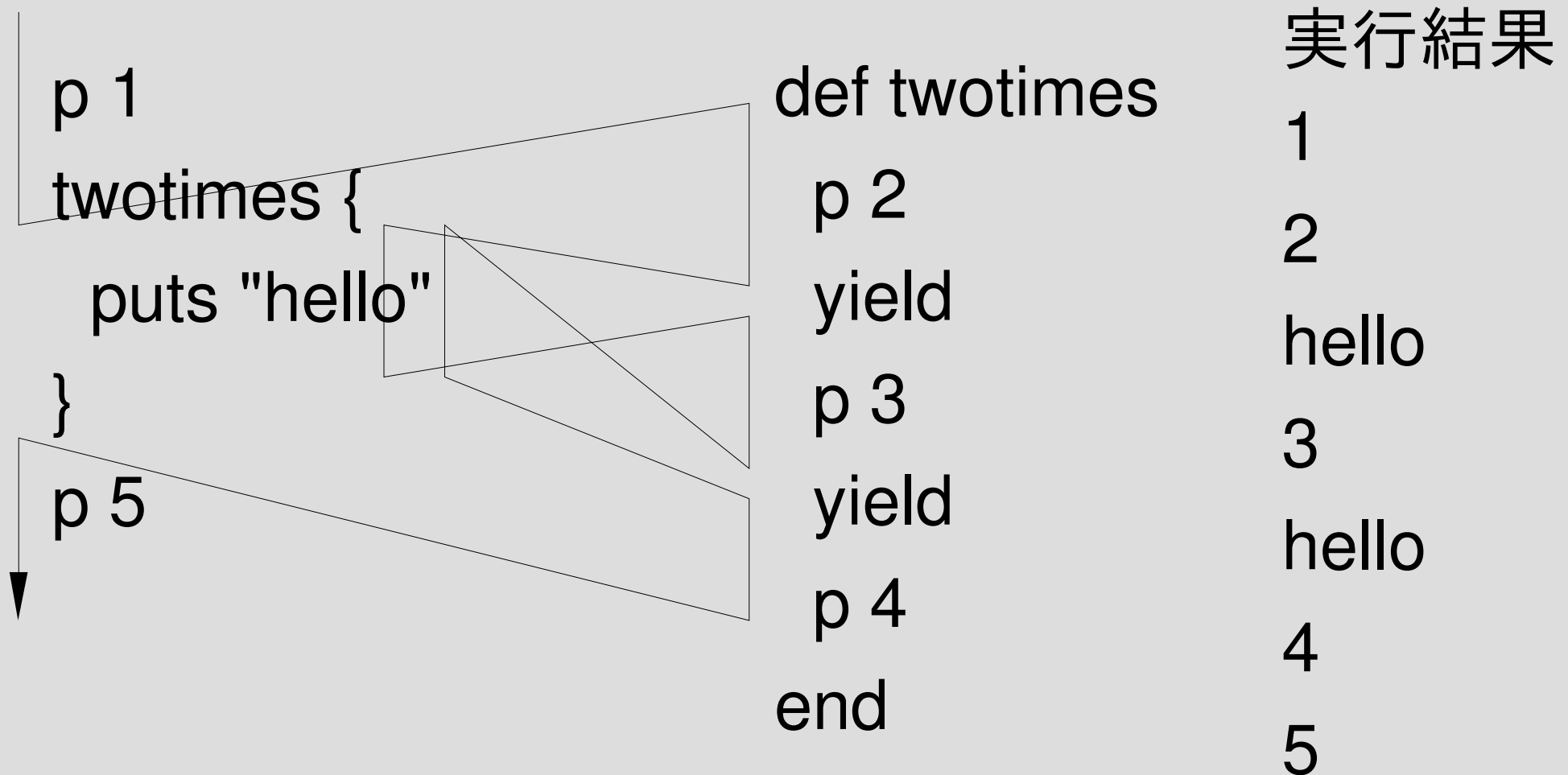

loop2 の制御の流れ

- yield はメソッドに与えられたブロックを呼び出す



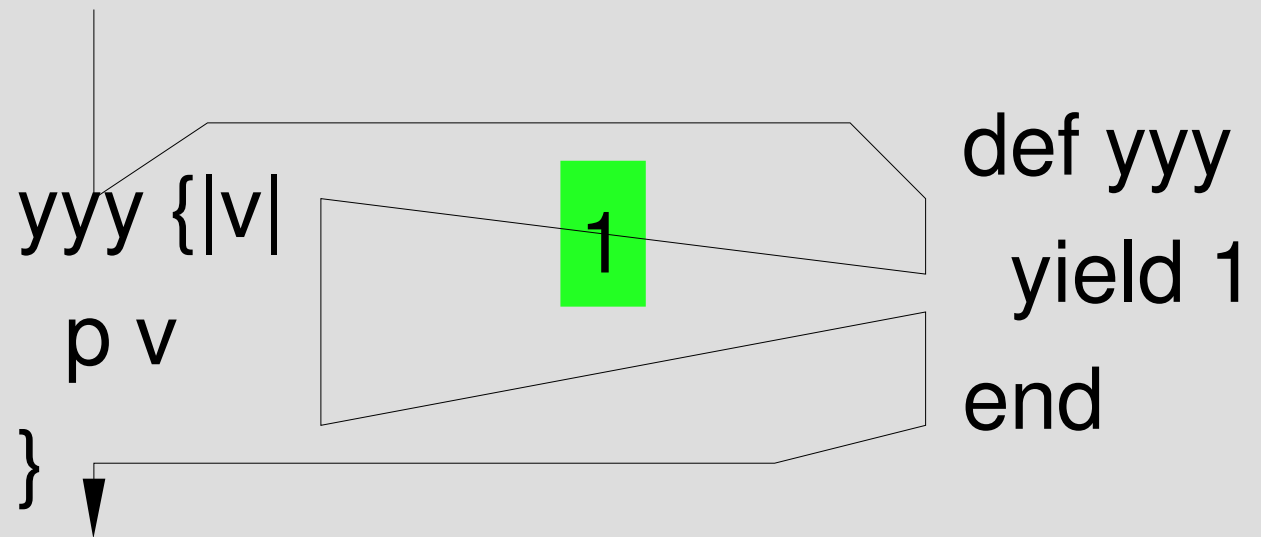
単純な例: twotimes

- yield はメソッドに与えられたブロックを呼び出す



yield の引数

- yield にはブロックへの引数を渡せる
yield arg
- ブロックはその引数を受け取れる
obj.meth {|arg| ... }



countup

- `countup(first, last) {|var| ... }`
- `first` から始めて `last` まで順に `var` に代入してブロックを実行する

```
def countup(first, last)
  i = first
  while i <= last
    yield i
    i += 1
  end
end
```

```
countup(3,5) {|v|
  p v
}
```

実行結果

3

4

5

countup の実行

- 3 から 5 まで

```
countup(3,5) {v|  
  p v  
}
```

```
def countup(first, last)
```

```
  i = first
```

```
  while i <= last
```

```
    yield i
```

```
    i += 1
```

```
  end
```

```
end
```

- 実行結果

3

4

5

Integer#upto

- countup に似たメソッドが標準で整数にある

```
3.upto(5) {|v|  
  p v  
}
```

- 実行結果
3
4
5

ブロックへの複数引数

- ブロックと `yield` は複数の引数をとれる

```
loop2dim(1,1) {|x,y|  
  p [x,y]  
}
```

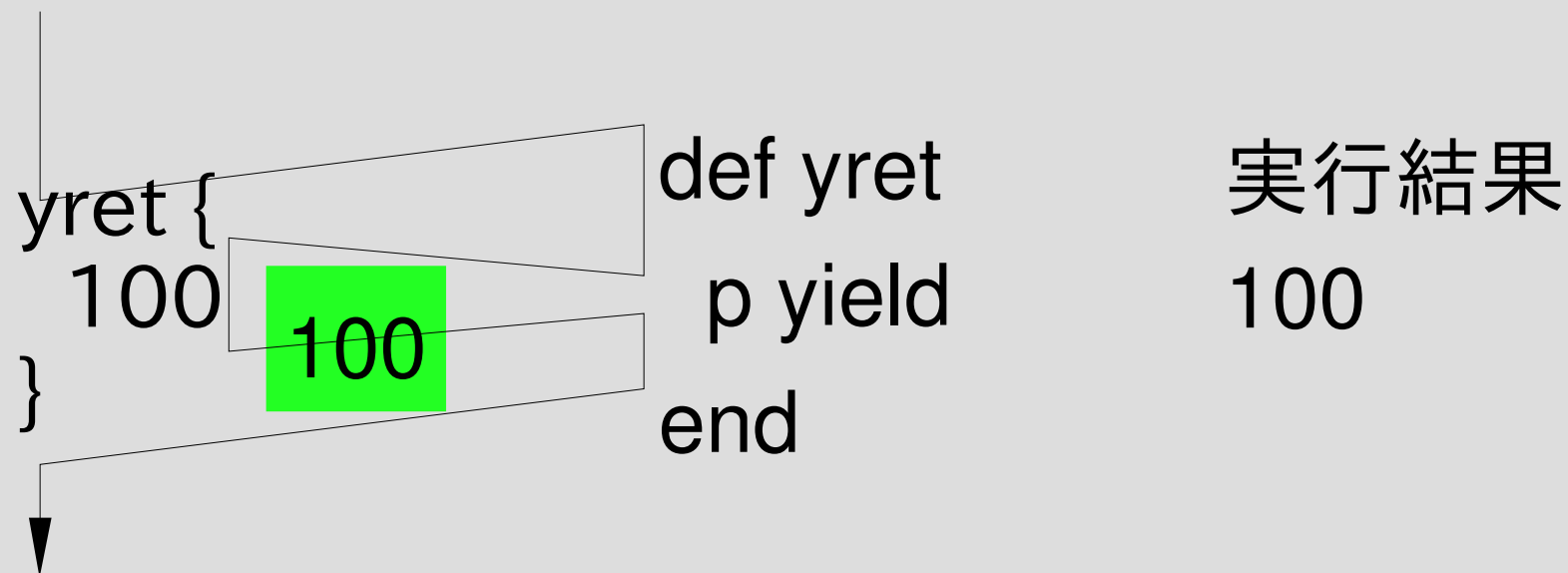
- 実行結果

```
[0,0]  
[0,1]  
[1,0]  
[1,1]
```

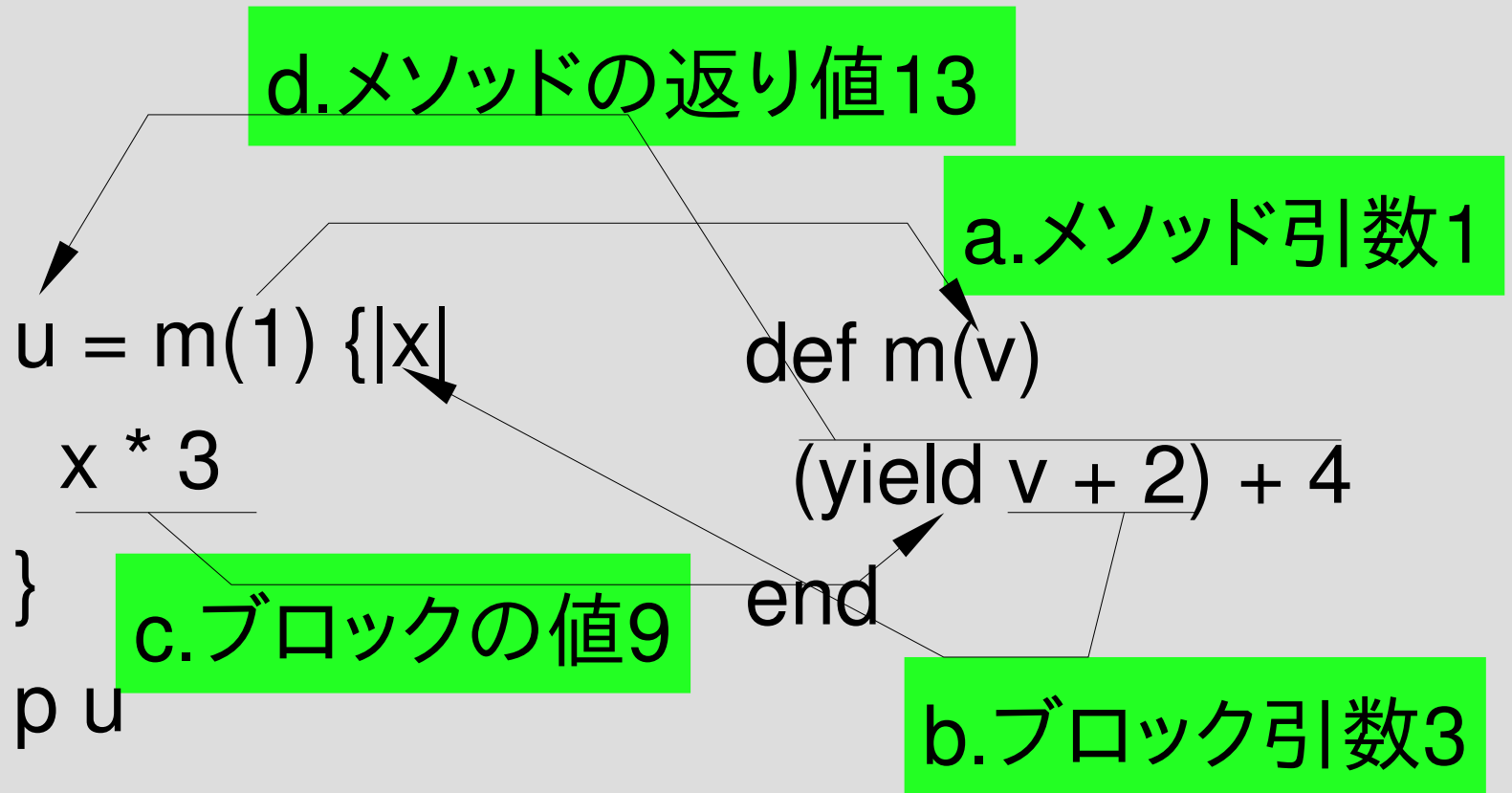
```
def loop2dim(xmax,ymax)  
  0.upto(xmax) {|x|  
    0.upto(ymax) {|y|  
      yield x, y  
    }  
  }  
end
```

yieldの返り値

- ブロック内の最後の文の値が yield の値になる



引数と返り値



実行結果

13

Array#map

- 配列の各要素にブロックを適用し、結果を集めた
あたらしい配列を生成

`[1,2,3].map {|v| v * 2 }` `#=> [2,4,6]`

Array#find_all

- 配列の各要素にブロックを適用し、結果が真だったものを集めた配列を生成

```
[1,2,3,4,5,6,7].find_all { |v| v % 3 == 0 }
```

#=> [3,6] 3の倍数だけが取り出される

Array#each

- 配列の各要素にブロックを適用する

```
[1,2,3].each {|v|
```

```
  p v  
}
```

実行結果

1

2

3

IO#each

- ファイルをオープンして得られる IO オブジェクトの each メソッド
- ファイルの各行にブロックを適用する

```
f = open("words")
f.each {|line|
  p line
}
f.close
```

実行結果

"\n"

"A\n"

"A's\n"

"AOL\n"

...

egrep.rb で IO#each を使う

```
pattern = ARGV.shift      # 第1引数の取り出し
filename = ARGV[0]       # 第2引数の取り出し
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                              # 正規表現オブジェクトに変換
f = open(filename)      # 第1引数のファイルをオープン
f.each {|line|          # ファイルの終わりまで一行ずつ読む
  print line if regexp =~ line # 行がマッチしたら表示
}
f.close                 # ファイルをクローズ
```

open もブロックを付けられる

- open にブロックをつけると、挙動が変わる
- ファイルオブジェクトを返り値にするのではなく、ブロックに渡す
- ブロックが終了したら、自動的に close される
呼出側で close しなくていい
- ブロックの返り値が open 自体の返り値になる

実行結果

```
open("words") { |f|  
  f.each { |line| p line }  
}
```

"\n"

"A\n"

"A's\n"

...

egrep.rb で open のブロックを使う

```
pattern = ARGV.shift      # 第1引数の取り出し
filename = ARGV[0]        # 第2引数の取り出し
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                                # 正規表現オブジェクトに変換
open(filename) { |f|      # 第1引数のファイルをオープン
  f.each { |line|         # ファイルの終わりまで一行ずつ読む
    print line if regexp =~ line # 行がマッチしたら表示
  }
}
```

ファイルをクローズ

File.foreach

- Fileクラスの foreach クラスメソッド
- ファイルをオープンして 1行ずつ読みこみ、各行にブロックを適用し、全部終わったらクローズする

```
File.foreach("words") {|line|
  p line
}
```

egrep.rb で File.foreachを使う

```
pattern = ARGV.shift      # 第1引数の取り出し
filename = ARGV[0]        # 第2引数の取り出し
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                                # 正規表現オブジェクトに変換
File.foreach(filename) { |line| # 第1引数のファイルを一行ずつ読む
  print line if regexp =~ line  # 行がマッチしたら表示
}
```

ARGF.each

- ARGF はコマンドライン引数に指定したファイルを示す IOオブジェクトもどき
- ARGVからファイル名を取り出す
- 複数のファイル名を指定すれば順に処理される
- ひとつも指定されなければ標準入力
- IOオブジェクトとほぼ同じメソッドがある

```
% ruby -e 'ARGF.each {|line| p line }' words
```

```
"¥n"
```

```
"A¥n"
```

```
...
```

egrep.rb で ARGF を使う

```
pattern = ARGV.shift      # 第1引数の取り出し
                          # ARGV の残りはファイル名
regexp = Regexp.compile(pattern) # 第2引数の文字列を
                                # 正規表現オブジェクトに変換
ARGF.each {|line|        # コマンドラインのファイルを一行ずつ読む
  print line if regexp =~ line # 行がマッチしたら表示
}
```

まとめ

- egrep もどきを Ruby っぽく書いてみた
10行→5行
- 特にブロックはいろいろ使える