

4.5 ネストした関数

関数型プログラミングスタイルでは、小さなヘルパー関数をたくさん作ることを推奨しています。先の例では、`sqrt`の実装はヘルパー関数 `sqrtIter`, `improve`, `isGoodEnough` を使っています。これらの関数の名前は `sqrt` の実装にだけ関係します。ふつう我々は、`sqrt` のユーザーがこれらの関数に直接アクセスすることを望みません。

そうすることを (そして名前空間の汚染を避けるのを)、ヘルパー関数を呼び出す関数自身の中へ入れることで強制できます。

```
def sqrt(x: Double) = {
  def sqrtIter(guess: Double, x: Double): Double =
    if (isGoodEnough(guess, x)) guess
    else sqrtIter(improve(guess, x), x)
  def improve(guess: Double, x: Double) =
    (guess + x / guess) / 2
  def isGoodEnough(guess: Double, x: Double) =
    abs(square(guess) - x) < 0.001
  sqrtIter(1.0, x)
}
```

このプログラムにおいて、中括弧 `{...}` は **ブロック** を囲みます。Scala のブロックはそれ自身が式です。各ブロックはその値を定義する結果式で終わります。結果式の前には補助的な定義があってもよく、それらはそのブロック内からしか見えません。

ブロック内の各定義は、後にセミコロンが続かねばならず、それによって後に続く定義や結果式と分離されます。しかし、次の条件の何れかが真でなければ、各行の最後にセミコロンが暗黙のうちに挿入されます。

- 1. 問題となる行がピリオドのような単語や、式の最後として正しくない中置演算子で終わる場合。
- 2. あるいは次の行が、式のはじまりとならないような単語で始まる場合。
- 3. あるいは括弧 (...) または、角括弧 [...] の内側にいる場合 (これらの中に複数の文を入れることはできません)。

したがって次は正しいです。

```
def f(x: Int) = x + 1;
f(1) + f(2)

def g1(x: Int) = x + 1
g(1) + g(2)

def g2(x: Int) = {x + 1}; /* ';' は必須 */ g2(1) + g2(2)
def h1(x) =
  x +
  y
  h1(1) * h1(2)

def h2(x: Int) = (
  x // 括弧は必須。
  + y // 括弧がないと 'x' の後にセミコロンが挿入される。
)
h2(1) / h2(2)
```

Scala は通常のプロック構造のスコープ規則を用いています。外側のブロックで定義された名前は、そこで再定義されない限り、内側のブロックからも見えます。この規則によって `sqrt` の例を簡略化できます。ネストさせた関数の追加パラメータとして `x` を渡す必要はありません。なぜなら外側の関数 `sqrt` のパラメータは常に見えるからです。次が簡略化されたコードです。

```
def sqrt(x: Double) = {
  def sqrtIter(guess: Double): Double =
    if (isGoodEnough(guess)) guess
    else sqrtIter(improve(guess))
  def improve(guess: Double) =
    (guess + x / guess) / 2
  def isGoodEnough(guess: Double) =
    abs(square(guess) - x) < 0.001
  sqrtIter(1.0)
}
```

- 「ギユ」は「行」のtypoでしょうか。あと、翻訳お疲れ様です。日本語でアクセスできるまとまったScalaの資料はまだ無いので、宮本さんの作業はたいへん素晴らしいものだと思います。 -- みずしま (2008-04-15 21:34:58)
- ご指摘ありがとうございます > みずしまさん。typo直しました。専門用語など心もとないと自覚があるので、随時ご指摘願えればと思います。 -- tmiya (2008-04-16 21:50:33)

名前:	<input type="text"/>
コメント:	<input type="text"/>