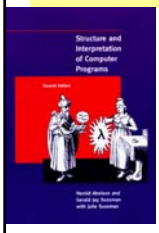
プログラミング言語(SICP)
3.Modularity, Objects, and State
3.1 Assignment and Local State

奥乃 博 (3章)・ 五十嵐 淳(4章)
大学院情報学研究科知能情報学専攻
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/12/ProgLang/
{okuno, igarashi}@i.kyoto-u.ac.jp

TAの居室は10号館4階奥乃1研, 2研, ソ基分野
糸原 連彦(M2) 奥乃研・音楽ロボットG
柳楽 浩平(M2) 奥乃研・ロボット聴覚G
福田 竜大(M1) ソ基分野(旧佐藤・五十嵐研)

NoStudent Left Behind

教科書



Structure and Interpretation of Computer Programs (SICP)

- 世界中のComputer Scienceのトップレベルの教科書(過去20年間)
- 1回生後期で前半を
- 2回生前期で後半を(湯淺先生)
- **MIT Press オンライン版**(無料)
- **Emacs Texinfo 形式**(無料)
- 日本語訳(邦訳・訳あり)約4.5K円

NoStudent Left Behind

教科書は持っているものとして進めます.

参考書とScheme 処理系

NoStudent Left Behind

1. **JAKLD Scheme(湯淺研開発・教育用計算機)**
   Java版(stand-alone, **携帯OK**), 他に Windows, Cygwin, Linux版あり.
   http://ryujin.kuis.kyoto-u.ac.jp/~yuasa/jakld/index-j.html
   Android版 (1回生の坂東君随意課題として作成)
   http://sigma-project.net/2011/01/25/androscheme/
2. **教育用計算機を使用. Install は不要.**
3. 他の処理系は**自己責任で使用**
4. ジョン・ベントリー(小林健一郎訳):『**珠玉のプログラミング―本質を見抜いたアルゴリズムとデータ構造**』(ピアソン)
   英語版を.
5. **世界中にSICPのサイト・コースウェア等あり**
6. **宿題は自分でやること(先に答えを見ない)**
7. **Plagiarism(剽窃)は不正行為!**

成績評価（上限は壱萬點）

1. 試験　　70%
2. 必修課題　30%
   - ① 宿題で出した練習問題．PDF をMail で提出
     翌週同日8時締切，
     ProgLang-1@zeus.kuis.kyoto-u.ac.jp
     File名：学籍番号-yourname-回数.pdf
   - ② 五十嵐先生分の課題
3. **加算システム：随意課題提出による“＋α”**
   - **① 第3・4章のすべての練習問題**
   - **② 論理回路シミュレータによる乗算の高速化**
   - **③ Twitterの並列オブジェクトによる記述**
   - **④ 銀行口座管理の並列オブジェクトによる記述**
   - **⑤ 他の学生の支援**

6

H24年度の目標　*No Student Left Behind*

落ちこぼれゼロ化作戦

3章はTA2名が担当の学生の合格率向上を競う．

Modulo (学籍番号の下1桁，2)

- **= 0　糸原君**
- **= 1　柳楽君**

7

LaTeX でレポートを書く

```
¥documentclass{a4paper,12pt}{article}
¥usepackage{listings}
¥title{タイトル} ¥author{学籍番号 氏名}
¥begin{document}
¥maketitle
¥section{反復型階乗}
¥lstset{numbers=left,basicstyle=¥small}
¥lstinputlisting{fact.scm}
  以上で，ファイル “fact.scm” に書かれたプログラムのリスティングが得
  られる．その下にプログラムの説明を書く．
    http://winnie.kuis.kyoto-u.ac.jp/~okuno/
      Lecture/11/IntroAlgDs/listing.tar.gz　にサンプルあり．
¥section{再帰型フィボナッチ数}
¥end{document}
```

PDFのファイル名：　学籍番号−名前−回数.pdf

例：　1029233333-HiroshiGOkuno-2.pdf

10

LaTeX でレポートを書く

NoStudent Left Behind

```
¥documentclass{a4paper,12pt}{article}
¥usepackage{listings}
¥begin{document}
¥section{階乗の定義}
¥begin{equation}
  n! =
    ¥left¥{
      ¥begin{array}{ll}
         1  & ¥mbox{if } n = 0 ¥¥
         n ¥times (n-1)! & ¥mbox{otherwise}
      ¥end{array}
    ¥right.
¥end{equation}
¥end{document}
```

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \times (n-1)! & \text{otherwise} \end{cases}$$

11

Jakld の 出力方法

NoStudent Left Behind

1. Command Prompt, cygwin で copy&paste
2. Shell の機能を使う
3. Output file を明示的に使用
   - (define out (open-output-file "outfile.txt"))
     out 同じファイル名のファイルがあると上書き
   - out
     #<port to outfile>
   - (display (fact 7) out) 結果を"outfile.txt"に書き込む
     5040
   - (newline out)
     #t
   - (close-output-port out)
4. call-with-output-file Output file の非明示的使用
   - (call-with-output-file "fact101.txt"
     (lambda (out)
     (newline out)
     (display (fact 101) out)
     (newline out)
     ))

12

レポート提出のプログラムコードに関する注意

- 処理系に通して動作を確認すること
- インデント(行頭の調整)を適切に行うこと

```
(define (make-withdraw balance)
  (lambda (amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds")))
```

```
(define (make-withdraw balance)
(lambda (amount)
(if (>= balance amount)
(begin (set! balance (- balance amount))
balance)
"Insufficient funds")))
```

4月11日・本日のメニュー

3.1 Assignment and Local State

3.1.1 Local State Variables

3.1.2 The Benefits of Introducing Assignment

3.1.3 The Costs of Introducing Assignment

第3章 モジュール化，オブジェクト，状態

第1章：手続き抽象化 primitive procedures

第2章：データ抽象化 primitive data

第3章：モジュール化 modularity 巨大なシステムを，自然な構成要素に分解して開発すること

局所化：開発・管理が容易，機能追加が容易

1．**オブジェクト(object-based approach)**
- How a computational object can change and yet maintain its identity
- **Environmental model** instead of substitution model of computation

2．**ストリーム(stream-processing approach)**
- Concurrent execution of programs
- **Delayed evaluation** to decouple simulated time in our mode from the order of events during evaluation

3.1 代入と局所状態(Assignment and Local State)

- オブジェクトが状態を持つ：振る舞いが履歴に依存
- 例：「100ドル引き出せるか？」
- 状態変数(例えば，残高)を使った実現

例：

1．**銀行の口座**：deposit, withdraw, balance
Debit, credit

2．**Twitterのアカウント**：tweet, follow, follower, favorite, retweet

状態変数 (*state variables*)

局所状態変数(*local state variables*)

3.1.1 局所的な状態変数

$100からスタート

(withdraw 25)
75
(withdraw 25)
50
(withdraw 60)
"Insufficient funds"
(withdraw 15)
35

非局所的な状態変数の使用

```
(define balance 100)
(define (withdraw amount)
  (if (>= balance amount)
      (begin (set! balance (- balance amount))
             balance )
      "Insufficient funds" ))
```

- **Assingment** (set! ⟨name⟩ ⟨new-value⟩)
- **順次実行** (begin $\langle exp_1 \rangle$ $\langle exp_2 \rangle$ … $\langle exp_n \rangle$)
  $\langle exp_n \rangle$ **の値を返す**

【Q】 上記の実現はどのような問題があるか.

局所的な状態変数の導入

```
(define new-withdraw
  (let ((balance 100))
    (lambda (amount)
      (if (>= balance amount)
          (begin (set! balance (- balance amount))
                 balance )
          "Insufficient funds" ))))
```

- **変数 balance は手続き new-withdraw に**
  カプセル化 (encapusulated)

Withdraw の抽象化

```
(define (make-withdraw balance)
  (lambda (amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        "Insufficient funds")))
(define W1 (make-withdraw 100))
(define W2 (make-withdraw 100))
(W1 50)
50
(W2 70)
30
(W2 40)
"Insufficient funds"
(W1 40)
10
```

口座ごとの状態変数

ペア(対、pair)を手続きで実現

NoStudent Left Behind

```
(define (cons x y)                    構築子
  (define (dispatch m)
    (cond ((= m 0) x)
          ((= m 1) y)
          (else (error "Argument not 0
                  or 1 -- CONS" m ))))
   dispatch)
(define (car z) (z 0))                選択子
(define (cdr z) (z 1))
```

- (define foo (cons 10 25))
- (car foo) ⇒ (foo 0) ⇒ 10
- (cdr foo) ⇒ (foo 1) ⇒ 25

22

Message passing

NoStudent Left Behind

```
(define (make-from-real-imag x y)
  (define (dispatch op)
    (cond ((eq? op 'real-part) x)
          ((eq? op 'imag-part) y)
          ((eq? op 'magnitude)
           (sqrt (+ (square x)
                    (square y) )))
          ((eq? op 'angle) (atan y x))
          (else
           (error "Unknown op –
            MAKE-FROM-REAL-IMAG" op))))
  dispatch )

(define (apply-generic op arg) (arg op))
```

23

銀行口座のオブジェクトによる定義

```
(define (make-account balance)
  (define (withdraw amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance )
        "Insufficient funds" ))

  (define (deposit amount)
    (set! balance (+ balance amount))
    balance )

  (define (dispatch m)
    (cond ((eq? m 'withdraw) withdraw)
          ((eq? m 'deposit) deposit)
          (else (error "Unknown request -- MAKE-ACCOUNT" m))))
  dispatch )
```

銀行口座の実行例

```
(define acc (make-account 100))

((acc 'withdraw) 50)
50

((acc 'withdraw) 60)
"Insufficient funds"

((acc 'deposit) 40)
90

((acc 'withdraw) 60)
30

(define acc2 (make-account 100))
```

全く別の口座が作成できる

3.1.2 The Benefits of Introducing Assignment

start with a given number x_1

x_2 = (rand-update x_1)

x_3 = (rand-update x_2)

...

普通のSchemeでは未定義

【Q】 上記の実装はどのような問題があるのか.

randの定義

```
(define rand
   (let ((x random-init))
      (lambda ()
         (set! x (rand-update x))
         x )))
```

【Q】 上記の実装はどのようなメリットがあるか.

証左：モンテカルロ法によるπの計算

任意の二つの整数が互いに素である確率は$6/\pi^2$

```
(define (estimate-pi trials)
   (sqrt (/ 6 (monte-carlo trials cesaro-test))))
(define (cesaro-test)
   (= (gcd (rand) (rand)) 1))
(define (monte-carlo trials experiment)
   (define (iter trials-remaining trials-passed)
      (cond ((= trials-remaining 0)
             (/ trials-passed trials))
            ((experiment)
             (iter (- trials-remaining 1)
                   (+ trials-passed 1) ))
            (else
             (iter (- trials-remaining 1)
                   trials-passed ))))
   (iter trials 0))
```

代入無しだと. . . Rand-update を使う

```
(define (estimate-pi trials)
   (sqrt (/ 6 (random-gcd-test trials random-init))))
(define (random-gcd-test trials initial-x)
   (define (iter trials-remaining trials-passed x)
      (let ((x1 (rand-update x)))
         (let ((x2 (rand-update x1)))
           (cond ((= trials-remaining 0)
                  (/ trials-passed trials))
                 ((= (gcd x1 x2) 1)
                  (iter (- trials-remaining 1)
                        (+ trials-passed 1)
                        x2))
                 (else
                  (iter (- trials-remaining 1)
                        trials-passed
                        x2 ))))))
   (iter trials 0 initial-x))
```

3.1.3 The Costs of Introducing Assignment

関数型プログラミング（Functional Programming）：代入なし

命令形プログラミング（Imperative Programming） 代入多用

簡単なプログラムで両者を比較

```
(define (make-simplified-withdraw balance)
   (lambda (amount)  ; 残高不足のチェック省略
      (set! balance (- balance amount))
      balance))

(define W (make-simplified-withdraw 25))
    (W 20)⇒ 5   (W 10) ⇒ -5
```

```
(define (make-decrementer balance)
   (lambda (amount)
      (- balance amount)))

(define D (make-decrementer 25))
    (D 20)⇒ 5   (D 10) ⇒ 15
```

置換えによる実行モデルで比較

```
(define (make-decrementer balance)
   (lambda (amount)
      (- balance amount)))
```

置換えによる実行モデル

```
((make-decrementer 25) 20)
⇒((lambda (amount) (- 25 amount)) 20)
⇒(- 25 20)
⇒ 5
```

【Q】 変数というのは何?

単なる「名前」ではない.

```
(define (make-simplified-withdraw balance)
   (lambda (amount)
      (set! balance (- balance amount))
      balance ))
```

置換えによる実行モデル

```
⇒((make-simplified-withdraw 25) 20)
⇒((lambda (amount)
     (set! balance (- 25 amount)) 25)
  20 )
⇒(set! balance (- 25 20)) 25
⇒ 25 ?
```

「値の格納場所」(place holder)

Sameness and Change（同等性）

【Q】次の両者は同じ(the same)？

```
(define D1 (make-decrementer 25))
(define D2 (make-decrementer 25))
```

【Q】次の両者は同じ(the same)？

```
(define W1 (make-simplified-withdraw 25))
(define W2 (make-simplified-withdraw 25))
      (W1 20) ⇒ 5
      (W1 20) ⇒ -15
      (W2 20) ⇒ 5
```

Even though `W1' and `W2' are "equal" in the sense that they are both created by evaluating the same expression,`(make-simplified-withdraw 25)', it is not true that `W1' could be substituted for `W2' in any expression without changing the result of evaluating the expression.

Sameness を天下りで決めないとChange は決められない

【Q】次の両者は同じ(the same)？

```
(define peter-acc (make-account 100))
(define paul-acc (make-account 100))
```

【Q】次の両者は同じ(the same)？

```
(define peter-acc (make-account 100))
(define paul-acc peter-acc)
```

別名(aliasing)

Pitfalls of Imperative Programming

```
(define (factorial n)
   (define (iter product counter)
      (if (> counter n)
          product
          (iter (* counter product) (+ counter 1))))
   (iter 1 1))
```

```
(define (factorial n)
   (let ((product 1)
         (counter 1))
     (define (iter)
        (if (> counter n)
            product
            (begin (set! product (* counter product))
                   (set! counter (+ counter 1))
                   (iter))))
     (iter)))
```

順序は大丈夫？

Exercise 3.7.

Consider the bank account objects created by make-account, with the password modification described in exercise 3.3. Suppose that our banking system requires the ability to make joint accounts. Define a procedure make-joint that accomplishes this. Make-joint should take three arguments. The first is a password-protected account. The second argument must match the password with which the account was defined in order for the make-joint operation to proceed. The third argument is a new password. Make-joint is to create an additional access to the original account using the new password. For example, if **peter-acc** is a bank account with password open-sesame, then

```
(define paul-acc
  (make-joint peter-acc 'open-sesame 'rosebud))
```

will allow one to make transactions on **peter-acc** using the name **paul-acc** and the password rosebud. You may wish to modify your solution to accommodate this new feature.

宿題：4月18日午前8時 締切

NoStudent Left Behind

1. 問題3.3, 3.4, 3.7
2. 実行例を添付すること
3. Program ファイルと説明をpdfで下記に送付

   ProgLang-1@zeus.kuis.kyoto-u.ac.jp

   file 名は 学籍番号-名前-1.pdf

- 友達に教えてもらったら，その人の名前を明記すること．Webは出展を明記．（otherwise『同じ』回答は減点）

DON'T PANIC!

45