コンピュータと教育　23－2
（１９９２．９．２５）

ＬＯＧＯを用いた「プログラミングの世界」への導入教育の経験

和田 勉

長野大学 産業情報学科

〒386-12　長野県上田市下之郷
Tel: (0268)38-2350
E-mail: ben@nagano.nagano.ac.jp

初心者の学部一年生対象の計算機・プログラミングの入門授業で行なっている、ＬＯＧＯを用いたコースの内容を紹介する。５年間改良を重ねてきて、現在では一応ほとんど誰でもこのコースをきちんと学べばプログラミングが理解できる、と言える状況になった。本コースではタートルグラフィックスを中心として、プログラミングの勉強だと意識せずスムーズに、ネスト、手続き、変数、ｉｆ、条件反復、再帰などのプログラミングの重要概念が理解できるように組み立ててある。

An experience of the introductory education to "Programming World" using LOGO

Ben Tsutom WADA

Division of Industry and Information Science, Nagano University

Shimonogo, Ueda-shi, NAGANO 386-12 JAPAN
Tel: (0268)38-2350
E-mail: ben@nagano.nagano.ac.jp

I introduce an introductory course of computers and programming using LOGO, which I have been teaching to novice undergraduate fresh-students. With my 5-years efforts of improvement, now I can say that almost anyone can understand the programming in studying this course properly. Mainly with turtle-graphics, this course is built so that students can smoothly understand the important issues of programming, such as nest, procedures, variables, if commands, conditional iteration, recursion, without conscious of being a study of computer progrmming.

１．はじめに

長野大学の産業情報学科は今年度(1992)で発足５年目になる。私は発足当初からこの学科の１年生（大部分は全くの初心者）に対する計算機・プログラミングへの入門の授業（一部のクラスのみ）を担当してきた。この中では発足当初から、初心者の学生を抵抗なく「プログラミングの世界」へ導くことを狙って、最初の段階の授業はＬＯＧＯを用いて行なってきた。５年間折りに触れそれなりの工夫を積み重ねてきて、今ではＬＯＧＯによるこのコースは「ほとんど誰でも、きちんとコースに沿って学べばプログラミングの主要概念が理解できる」と言えるところまで一応持ってくることができた、と自己評価している。

このＬＯＧＯのコースに関しては、私のもう一つの研究課題である「計算機教育用映像システム」との関連でこれまでも折りにふれて何回か発表してきた。([1]-[5]) 今回はコースの組み立て方自体の紹介を主にして報告する。

２．授業とコースについて

このＬＯＧＯによるコースは、産業情報学科の新入生がクラス分けされて受ける１年生対象の必修授業の私の担当クラスで実施している。この授業は週一回３時間の通年授業で、年間の授業回数は２１～２３回程度になる。年間の授業時間の割り振りは以下のようにしている。

０．（第１週の前半）ガイダンスとビデオ（ＴＶ番組）の上映
１．（第１週後半と第２～８週）ＬＯＧＯのコース
２．（第９週からの３回）ＵＮＩＸの基本的操作
３．（その後年度末までの１０～１２回）Ｃによるプログラミング、その他

上記の１が今回報告するＬＯＧＯのコースで、上記のように第１回のみ１時間半でその後は各回３時間、計３時間×７.５回となっている。各回で扱っている内容の表題を以下にまとめて示す。より詳しい内容は第３節で紹介する。

第１回．電源投入とＬＯＧＯの立ち上げ
(0.5回)　タートルグラフィックスの基本的概念
　　タートルグラフィックスの基本的な命令
　　（forward, left等）
　　repeat命令
第２回．repeat命令のネスト
　　（引数なしの）手続き
第３回．引数を持つ手続き
　　複数個の引数を持つ手続き
第４回．変数
　　手続き中からの手続き呼出し
第５回．条件判定（ｉｆ）
第６回．ｉｆのネストとインデンテーション
第７回．条件反復（ｗｈｉｌｅ）
　　再帰
第８回．自由課題

本コースは、中身はもちろんプログラミングの勉強（ないしはその基礎となる「論理的思考のトレーニング」）なのだが、最初から「これからコンピュータのプログラミングを勉強します」などとは言わず、ただ図を描いてパズル的な頭脳遊戯をしているだけの「ふり」をする。それにより学生達が心理的な抵抗なしに「プログラミングの世界」へ入ってゆけるよう狙っており、ＬＯＧＯ（のタートルグラフィックス）をそのための道具（「砂糖」）として使っている。

本コースの冒頭の部分は以下のように始まる：

（まず電源投入とＬＯＧＯの立ち上げの方法を教え、各自起動させる。）
「画面中央に三角形のマークがあります。これがタートル、つまり亀です。」
「ここでキーボードから"forward 100"と入れてみます。するとこういう画面になります。亀が１００歩前進したわけです。するとこの亀はペンを持っていて、動くとその跡がこのように線になって残ります。これでこれからいろいろな図を描いてみようというわけです。」
（ここでforward（前進）に加えて、back（後退）、left（その場で左回転）、right（その場で右回転）等の命令を紹介する。）
「これで正方形が描けるわけです。」（実際にforwardとrightを４回繰り返して描いてみせる。）
「しかしこれではめんどうです。今やったのを見ると、同じ命令を４回繰り返しています。こういう時は時はrepeatという「省略記法」が使えます…」

以後も本コースはおよそ同じような調子で進めてゆく。かなり後まで「プログラミング」だとは言わない。もちろん多少ともプログラミングの経験のある者にはすぐ分かってしまうが。

上の例にも出てきた「今ある何か『めんどくさいこと』を解決する『道具』を導入する」ということが、コースを進める上で大きな「とって」の一つになっている。上記の例以外にも、
・手続き（一度だけ登録（定義）しておけばその名前を入れるだけで本体を何度でも実行してくれる

道具）、
・引数（例えば任意の大きさの正方形を描くのに一つの手続きだけで済むようにするための道具）
・変数（forward 5 right 90 forward 10 right 90 ...（四角の渦巻き模様を描く）などという例でもrepeatによる省略ができるようにする道具）
などを使っている。

なお、このコースに対する「ＡＶ講義録」（以前の授業を記録したビデオテープ）と配布用プリントは全てすでに揃えてあり、授業はこれらを使って進めている。これらは毎年一部は作り／書きなおしている。

３．コース内容

－－－－－－－－－－－第１回－－－－－－－－－－－

＊電源投入とＬＯＧＯの立ち上げ

＊タートルグラフィックスの基本的概念

「画面中央に見えているのが「タートル（亀）」である。」「"forward 100"と入れてみると亀が１００歩前進する。」「この亀はペンを持っていて、動くとその跡が線になって残る。」

＊タートルグラフィックスの基本的な命令

forward（前進）に加えて、back（後退）、left（その場で左回転）、right（その場で右回転）等の命令を紹介する。

forward 50とright 90を４回繰り返して正方形を描いてみせる。

「しかしこれではめんどうである。そこで…（次項へ）」

＊repeat命令

「同じ命令を繰り返す時はrepeatという「省略方法」が使える。repeat 回数 [繰り返すべき命令] と書けばよい。従ってこの場合はrepeat 4 [forward 50 right 90] と入力するだけで正方形を描くことができる。」

三角形についても同様に例をやってみせる。

演習出題: 正六角形を描くなどいろいろ試してみよ。

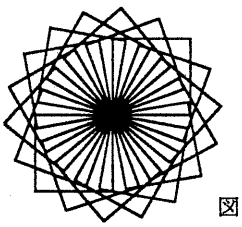


図１

－－－－－－－－－－－第２回－－－－－－－－－－－

＊repeat命令のネスト

図１（多数の正方形による花模様）をまず

```
repeat 4 [forward 50 right 90]
left 20
repeat 4 [forward 50 right 90]
left 20
  ...
```

と繰り返して描いてみせる。

「一応描けるが、しかしこれでは手数がかかる。」

「repeat命令の一般型は"repeat 回数 [命令の並び]"である。『命令の並び』には任意の命令が書けるので、当然別のrepeat命令が入ってもよい。」

「これを使えば図１は

repeat 回数 [repeat 4 [forward 50 right 90] left 20]

で書ける。（回数をいくつにすればいいかは伏せておき演習問題とする）」

演習出題：提示したいくつかの図形を描いてみよ。

（引用注：提示した図形には一重のrepeatで描ける図形とrepeatのネストが必要な図形が混在している。）

＊（引数なしの）手続き

「命令に名前を付けて登録することができる。これを手続きと言う。」

「例えばまず

```
to seihoukei
repeat 4 [forward 80 right 90]
end
```

と入れる。この時は画面に何も描かれないが、これで手続きseihoukeiが登録された。」

「この後

```
seihoukei
```

と入れると、それだけでたちまち正方形が描かれる。（デモしながら説明）」

（引用注：理解しやすくするため手続きの定義のことを「登録」とも呼んでいる。）

「手続きは単純な呼び出し方だけではなくプリミティブと同様に使える。」

「例えば

```
repeat 36 [seihoukei right 10]
```

などという使い方もできる。これは

```
repeat 36 [repeat 4 [forward 80 right 90] right 10]
```

と同じ動作をする。」

「手続きの本体は複数行でもよい。その場合先頭から順に実行される。」

例：

```
to seisankaku
right 30
repeat 3 [forward 80 right 120]
end
```

－－－－－－－－－－第３回－－－－－－－－－－

＊引数を持つ手続き

前回例示した引数なしの手続きでは、常に一辺８０の正方形しか描けない。

```
to seihoukei
repeat 4 [forward 80 right 90]
end
```

「引数付きの手続き」を使えば以下のように「任意の大きさの正方形を描く手続き」を作ることができる。

```
to seihoukei1 :hen
repeat 4 [forward :hen right 90]
end
```

「引数なしの手続きはseihoukeiを呼出すには単に

```
seihoukei
```

とした。これに対し、引数付きの手続きは例えば

```
seihoukei1 50
```

と引数を付けて呼出す。これにより引数部分が置きかわり

```
repeat 4 [forward 50 right 90]
```

と同じ動作が起こる。」

「引数を持つ手続きは呼ぶ時も必ず引数を付けて呼ばなければならない。」

「引数は手続き本体中に一回だけでなく複数回現れてもよい。」

例：

```
to juuji :ude
repeat 4 [forward :ude back :ude right 90]
end
```

演習出題：図２を描く引数なしの手続きを作れ（辺長６０に固定）

同 引数付きの手続きを作れ（辺長は引数で指定）

＊複数個の引数を持つ手続き

「引数は複数個使うことができる。例えば長方形を描く手続きを以下のように作ることができる。」

```
to box :tate :yoko
forward :tate
right 90
forward :yoko
right 90
forward :tate
right 90
forward :yoko
right 90
end
```

「呼出す際は例えばbox 20 50とする。第１引数は第１引数どうし、第２引数は第２引数どうしがくっつき、それぞれ置き換えが起こる。この例では以下のようになる。」

```
forward 20
right 90
forward 50
right 90
forward 20
right 90
forward 50
right 90
```

「引数２つの手続きは必ず引数２つ付けて呼出す。box 30やbox 40 15 80は間違い。」

「同様に引数を３個、４個…持つ手続きも使える。」

演習出題: 第１引数が３の場合は図３－１、４の場合は図３－２、５の場合は図３－３…と描き分ける手続きを作れ。辺長は第２引数で指定するようにせよ。

ヒント：forward 30 + 10 や forward :abc * 5 のように数式を使うことができる。例えば repeat :kai [forward 100 / :kai wait 60] などと書くこともできる。

補題１:まず図３－１～３を描く手続きをそれぞれ独立に描いてみよ。

補題２:正ｎ角形を描く手続き（ｎは引数で指定）を書いてみよ。

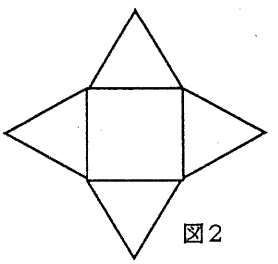


図２

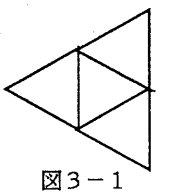


図３－１

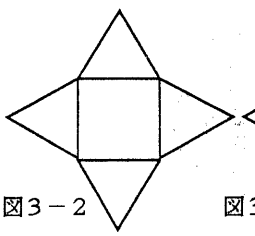


図３－２

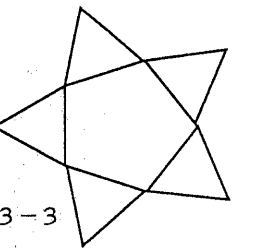


図３－３

－－－－－－－－－－第４回－－－－－－－－－－

＊変数

「図４を描く。今までの知識では以下のようになる。」

```
forward 5
right 90
forward 10
right 90
forward 15
right 90
...
forward 95
right 90
```

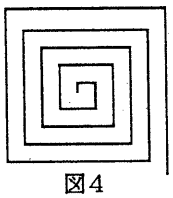


図４

「しかしこれではめんどうである。『変数』を使うと以下のように書ける。」

```
make "nagasa 0
repeat 19 [make "nagasa :nagasa+5 forward :nagasa right 90]
```

（引用注:「make〝変数 式 」は「変数に式の値を代入」の意。「：」は変数の値の参照の意。）

「変数とは…

『記憶場所』である。

常に一つの数値が覚えておかれている。

いつでも「参照」や「変更」ができる。

最初はどんな値が入っているか分からない。

一度「変更」するとその値で置きかわる。

「参照」するとその値が使える。

何度参照されても値は変わらない。

再度「変更」すると古い値は消えて置きかわり、

以後の参照ではその値が出てくる。

変数名は任意。

名前を違えれば複数個の変数が使える」。

「make〝nagasa :nagasa+5では『変数をまず参照し次いでその同じ変数を変更』している」

演習出題：図４を逆に外側から内側に向かって描く手続き。

＊手続き中からの手続き呼出し

まず正方形を描く手続きseihoukeiを作る。

「それを使い図５を描く手続きを以下のように作ることができる。」

```
to mseihoukei
make "naga 0
repeat 20 [make "naga :naga+5 seihoukei :naga]
end
```

「このように手続き中から他の手続きを呼出すことができる。」「呼出される手続きは、呼出す手続きの実行時に登録されていないとエラーとなる。（登録時ではなく。）」

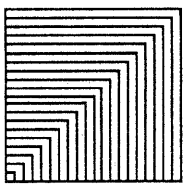


図５

「手続きmseihoukeiをさらに呼出す手続きrepmseiを作ってみる。」

```
to repmsei
repeat 10 [mseihoukei clearscreen]
end
```

「試しにこれを『手続き中からの手続き呼出し』を使わずにいきなり書き下したらどうなるだろうか。」

```
repeat 10 [make "naga 0 repeat 20 [make "naga
:naga+5 repeat 4 [forward :naga right 90] ]
clearscreen]
```

「このように非常に分かりにくくなる。プログラミングでは、扱う問題が複雑になればなるほどこのような『階層的』な方法をとることが重要になる。」（引用注：ここで初めて「実は今まで知らず知らずのうちにプログラミングの勉強をしていたのだ」と明言する。といってもすでにそれは分かっている学生も多いが。）

課題出題：正三角形を描く手続きを作り、すでに作った図２を描く手続き（引数なし、引数付きとも）と図３－１～３を描く手続きを、それを呼出して描くように作りなおせ。

アドバイス：「正三角形を描くプリミティブがあると思って描け」

－－－－－－－－－－第５回－－－－－－－－－－

＊条件判定（ｉｆ）

正八角形を描く手続きを例示する。

```
to hachikaku :hen
repeat 8 [fd :hen right 45]
end
```

これを「引数が適当な（画面をはみ出さない）値の場合のみ描く」ように改良する例を使い（ｅｌｓｅなしの）ｉｆを解説する。

```
to chachikaku :hen
if :hen <= 100 [repeat 8 [forward :hen right 45]]
end
```

「一般にIF命令は次のように書くことができる。」

IF　条件　[命令]

次いで「引数が大きすぎる場合は八角形に代えて正方形を描く手続き」を例示し、ｅｌｓｅのあるｉｆを解説する。

```
to hachiyon :hen
if :hen <= 100 [repeat 8 [fd :hen rt 45]]
               [repeat 4 [fd :hen rt 90]]
end
```

「この形のIF命令は一般に次のような形になる。」

IF　条件　[命令-1]　[命令-2]

「ｔｈｅｎ部、ｅｌｓｅ部とも複数個の命令を並べることもできる。」

課題出題：正三角形・正方形・正六角形を以下のように描きわける手続きを書け。
引数で指定された一辺の長さが
３０未満の場合は　　正六角形だけを
３０以上５０以下の場合は　正六角形と正方形を
５０を越える場合は　　正三角形と正方形を

引用注：この問題はｉｆのネストもand, orも使わずに書けるようにしてあることに注意。標準的な回答は

```
to sanyonrok :hen
if :hen <= 50 [正六角形を描く]
              [（不成立なら）正三角形を描く]
if :hen >= 30 [正方形を描く]
end
```

といったものになる。なお

```
to sanyonrok :hen
if :hen <= 50 [正六角形を描く]
if :hen > 50  [正三角形を描く]
if :hen >= 30 [正方形を描く]
end
```

といった回答をした者には「一応良いがｉｆ２つでも書けるのでそう書きかえてみるように」と助言している。また

```
if :hen >= 30 <= 50
```

あるいは

```
if 30 <= :hen <= 50
```

などと書こうとする者も居るので「本当にそういう判定が必要な時はandというものを使うがこの問題には必要ない。問題文を読みなおせ。」と助言している。

－－－－－－－－－－－第６回－－－－－－－－－－－

＊ｉｆのネストとインデンテーション

例題提示：
引数の値が偶数の場合
　・引数の値が100未満なら左側に正六角形を描く
　・100以上なら左側に正方形を描く
引数の値が奇数の場合
　・引数の値が100未満なら右側に正六角形を描く
　・100以上なら右側に正方形を描く

「これは４通りの場合分けが必要なのでｉｆ命令１つだけでは書けない。」
「ｉｆ命令も命令の一種なので、ｉｆ命令中に他のｉｆ命令があってもよい。これを利用するとこの例題は以下のように書ける。」

```
to sayua :hen
if ( remainder :hen 2 ) = 0 [ if :hen < 100 [
repeat 6 [forward :hen left 60] ] [ repeat 4
[forward :hen left 90] ] ] [ if :hen < 100 [
repeat 6 [forward :hen right 60 ] ] [ repeat 4
[forward :hen right 90 ] ] ]
end
```

「これは正しく動作はするが見にくい。これを見やすくしたものを以下に示す：」

```
to sayua :hen
if ( remainder :hen 2 ) = 0
        [
        if :hen < 100
                [ repeat 6 [forward :hen left 60] ]
                [ repeat 4 [forward :hen left 90] ]
        ]
        [
        if :hen < 100
                [ repeat 6 [forward :hen right 60 ] ]
                [ repeat 4 [forward :hen right 90 ] ]
        ]
end
```

「このような書き方をインデンテーション（字下げ）という。」

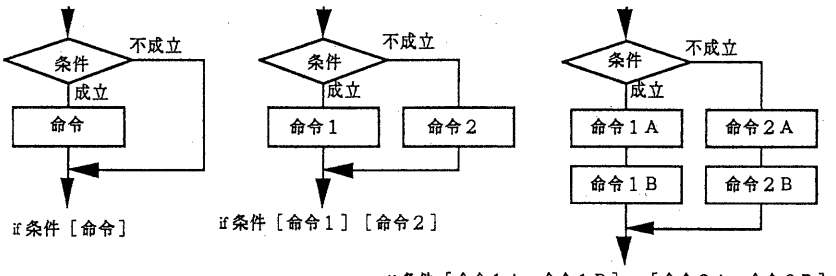


図６

前の例題の変形を提示（「どちらの場合も…」という句を前の例題に２つ追加したもの）：
引数の値が偶数の場合
　・引数の値が100未満なら左側に正六角形を描く
　・100以上なら左側に正方形を描く
　どちらの場合もその後左側に正三角形を描く
引数の値が奇数の場合
　・引数の値が100未満なら右側に正六角形を描く
　・100以上なら右側に正方形を描く
　どちらの場合もその後右側に正三角形を描く

```
to sayub :hen
if ( remainder :hen 2 ) = 0
          [
          if :hen < 100
                    [ repeat 6 [forward :hen left 60] ]
                    [ repeat 4 [forward :hen left 90] ]
          repeat 3 [forward :hen left 120]
          ]
          [
          if :hen < 100
                    [ repeat 6 [forward :hen right 60 ] ]
                    [ repeat 4 [forward :hen right 90 ] ]
          repeat 3 [forward :hen right 120]
          ]
end
```

課題出題：上の手続きSAYUBを変更して、以下の動作をする手続きSAYUC, SAYUD, SAYUEをそれぞれ作れ。新たに命令を付け加えることはせずに、SAYUB内の各部分を組み替え書き替えて作ることを考えよ。できるだけ規則的なインデンテーションをつけよ。

手続きSAYUC：
引数の値が偶数の場合
　・引数の値が100未満なら左側に正六角形を描く
　・100以上なら左側に正方形を描く
　どちらの場合もその後左側に正三角形を描く
引数の値が奇数の場合
　・引数の値が80未満なら右側に正六角形を描く
　・80以上なら右側に正方形を描く
　どちらの場合もその後右側に正三角形を描く
（引用注：sayucはかなり高率の正解がある。）

手続きSAYUD：
引数の値が偶数の場合
　・引数の値が100未満なら左側に正六角形と
　　正三角形を描く
　・100以上なら左側に正方形を描く
引数の値が奇数の場合
　・引数の値が100未満なら右側に正六角形を描く
　・100以上なら右側に正方形と正三角形を描く
（引用注：sayudには[ ]の位置・個数がおかしい回答がときおり見うけられる。）

手続きSAYUE：
引数の値が偶数の場合
　・引数の値が100未満なら左側に正六角形を描く
　・100以上なら左側に正方形を描く
引数の値が奇数の場合
　・引数の値が100未満なら右側に正六角形を描く
　・100以上なら右側に正方形を描く
４つのいずれの場合でもその後左側と右側に
それぞれ正三角形を描く
（引用注：sayueに以下のような回答をする者がいくらかいる。

```
to sayue :hen
if ( remainder :hen 2 ) = 0
          [
          if :hen < 100
                    [ repeat 6 [forward :hen left 60] ]
                    [ repeat 4 [forward :hen left 90] ]
          repeat 3 [forward :hen left 120]
          repeat 3 [forward :hen right 120]
          ]
          [
          if :hen < 100
                    [ repeat 6 [forward :hen right 60 ] ]
                    [ repeat 4 [forward :hen right 90 ] ]
          repeat 3 [forward :hen left 120]
          repeat 3 [forward :hen right 120]
          ]
end
```

これに対しては「命令数がsayubに比べて増えている」ことを指摘して再度改良するよう指導する。）

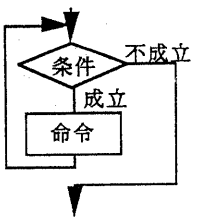
－－－－－－－－－－第７回－－－－－－－－－－
＊条件反復（ｗｈｉｌｅ）
例題提示：
まず一辺が引数で与えられた長さの正六角形を描く。
次に一辺がその1/2の正六角形を描く。
次に一辺がさらにその1/2の正六角形を描く。
以下同様に一辺の長さが５未満になるまで（５以上であるかぎり）続ける。

```
to touhi :ihen
make "chen :ihen
while [:chen >= 5]
        [
        repeat 6 [forward :chen right 60]
        make "chen :chen / 2
        ]
end
```

「ここでは：
まず変数CHENの値が５以上か調べ、
　５未満（不成立）の場合は終了。
５以上（成立）の場合は
　REPEAT 6 [FORWARD :CHEN RIGHT 60]を実行。
変数CHENの値を1/2にする。
再び変数CHENの値が５以上か調べ、
　５未満の場合は終了。
５以上の場合は再び
　REPEAT 6 [FORWARD :CHEN RIGHT 60]を実行。
変数CHENの値をさらに1/2にする。
再び変数CHENの値が５以上か調べ、
　５未満の場合は終了。
　　：
と、変数CHENの値が５未満（不成立）になるまで何回でも続けている。」

「一般にWHILE命令では：
まず「条件」が調べられ、
それが成立しない場合はすぐに次の命令へ行く。
条件が成立した場合は「命令」が実行される。
また「条件」が調べられ、
成立しなくなっている場合は次の命令へ行く。
成立した場合はまた「命令」が実行される。
また「条件」が調べられ、
成立しなくなっている場合は次の命令へ行く。
ということが繰り返される。」

演習出題：例題とは逆に次々に倍々に大きい六角形を描く手続きを作れ。（終了条件は画面の大きさに合わせて適当に。）



ｗｈｉｌｅ　［条件］　［命令］

図7

＊再帰（引用注：本節は1992年度の新作）
「本章は急がなくていいから納得いくまで考えること。」
「以下の手続きTOUHIRは上記の例（whileの例題）TOUHIと全く同じ動作をする。試してみてなぜそうなるか考えてみよ。

```
to touhir :ihen
if :ihen >= 5
    [
    repeat 6 [forward :ihen right 60]
    touhir :ihen / 2
    ]
end
```

理解できたら次は下記の手続きTOUHIEAを登録せよ。

```
to touhiea :ahen
if :ahen >= 5
        [
        repeat 6 [forward :ahen right 60]
        touhieb :ahen / 2
        penerase
        repeat 6 [forward :ahen right 60]
        pendown
        ]
end
```

TOUHIEA中では手続TOUHIEBを呼び出している。TOUHIEBの内容は以下の通りTOUHIEAとほぼ同じ内容で登録せよ。

```
to touhieb :bhen
if :bhen >= 5
        [
        repeat 6 [forward :bhen right 60]
        touhiec :bhen / 2
        penerase
        repeat 6 [forward :bhen right 60]
        pendown
        ]
end
```

TOUHIEB中では手続TOUHIECを呼び出している。TOUHIECも以下のように内容はほぼ同じで登録せよ

```
to touhiec :chen
if :chen >= 5
        [
        repeat 6 [forward :chen right 60]
        touhied :chen / 2
        penerase
        repeat 6 [forward :chen right 60]
        pendown
        ]
end
```

TOUHIC中では手続TOUHIEDを呼び出しているがこれはひとまず無視して登録しないでおく。さてここまで登録して

```
TOUHIEA 5
```

と呼び出すとどんな動作をするだろうか？　考えて予想をたてたあと試してみよ。また

```
TOUHIEA 10
```

ではどんな動作をするだろうか？　同様に予想をたてたあと試してみよ。予想と動作が違っていたら、どこに考え違いがあったか納得が行くまで考えてから次に進むこと。」

（引用注：peneraseはタートルが持つペンを「消しゴム」に持ちかえる命令。消しゴムを持ったタートルが線の上をなぞると線は消える。再度ペンに持ちかえる命令はpendown。）

「次に、以下の手続を登録してから、上と同様に TOUHIE 5　あるいは　TOUHIE 10 と呼び出してみよ。どんな動作が起こるだろうか？　また TOUHIE 20　や TOUHIE 40　ではどうなるだろうか？　これはどうなっているのだろうか？

```
to touhie :ihen
if :ihen >= 5
    [
    repeat 6 [forward :ihen right 60]
    touhie :ihen / 2
    penerase
    repeat 6 [forward :ihen right 60]
    pendown
    ]
end
```

ここまで理解できたら、振り返って最初の手続TOUHIRを見てみよ。間違った理解をしていなかったろうか？

このような方法を、REPEATやWHILE を使った繰り返し（反復(Iteration)と呼ぶ）に対し「再帰(Recursion)」と呼ぶ。」

「上記の例がよく理解できてさらに余裕がある者は、次の例を考えてみよ。

```
to eda :enag
if :enag >= 5
    [
    forward :enag
    eda :enag / 2
    back :enag
    right 90  forward :enag  left 90
    eda :enag / 2
    right 90   back :enag    left 90
    ]
end
```

これを例えば

EDA 5　と呼び出したらどんな動作をするだろうか？ EDA 10　や　EDA 20　ではどうだろうか？　前の例と同様に、予想をたてたあと試してみよ。前と同様に、以下のものと対比して考えるとよい。

```
to edaa :anag
if :anag >= 5
    [
    forward :anag
    edab :anag / 2
    back :anag
    right 90  forward :anag  left 90
    edac :anag / 2
    right 90   back :anag    left 90

    ]
end

to edab :bnag
if :bnag >= 5
    [
    forward :bnag
    edad :bnag / 2
    back :bnag
    right 90  forward :bnag  left 90
    edae :bnag / 2
    right 90   back :bnag    left 90

    ]
end

to edac :cnag
（一部引用省略）
end
```

（以下EDAD, EDAE,...は省略）

予想は当たったろうか？　当たらなかったらもう一度考え直してみよ。」

（引用注：本節は1992年度の書き下ろしである。再帰を理解させるのには「同じ内容の手続きがたくさんありそれを順番に呼び出すのだ」と考えると分かりやすいのではないか、とは以前から個別指導などを通じて感じていた。教えてみた結果は、一応分かった者と「何をどう分かればいいのかよく分からない」という様子の者がほぼ半々、というところであった。）

－－－－－－－－－－－－第８回－－－－－－－－－－－－
＊自由課題（引用注：第７回に下記の課題文を提示して第８回を全部演習時間としている。）
「各自それぞれ、自由な図形を描く、独創的な手続きを作り、その手続き、およびそれが描く図形の略図をレポートとして提出せよ。（他の手続きを呼び出している場合は、その呼び出されている手続きも全て書くこと。）授業中に例として説明した手続きを、「呼び出される手続き」としてそのまま使っても差しつかえない。

単に「奇麗な」図形を描くことが目的ではない。単純な「長ったらしい」手続きで「奇麗な」図形を描いてみても意味がない。これまでに学んだことを生かした作品を期待する。」

４．自己評価

実施のたびに少しずつ改良を重ねてきた本コースは、幸いなことに「第１回から順を追って身を入れて学べば大部分の学生は、その内容を、そしてそこに『砂糖』にくるまれてちりばめてあるプログラミングの諸概念をおおむね理解できる」という状況まで持ってくることができた、と自己評価している。

昨年までの経験では、本コースの目下の最大の問題は、本コースを終了して（潜在的には）すでに基本的なプログラミングが理解できていると思われるのに、その学生達がその後の「普通の」プログラミング言語（Ｃなど）でのプログラミングが必ずしもスムーズに理解できるとは言えない、ということである。ただ今年(1992)からＣによるプログラミングも続けて私が担当する予定（昨年までは他の教員が担当していた）なので、ＬＯＧＯで出したのと類似の例題を出して対比づけを容易にするなどの工夫を今まで以上にしてみるつもりであり、それによりこの点はかなり解決できると踏んでいる。

もう一つ、欠席者の問題がある。いわゆる「文化系」の大学だからだろうか、「第一回から段階を追って学ばなければならない、途中を飛ばして後の段階を学ぶことはできない」という認識がないらしく、何回か欠席した後突然出席するなどという学生がかなり居る。当然内容が理解できるはずもないが、それを「段階を踏んでいないのだからあたりまえ」とは考えず、「自分はこの分野に向いていない」とか「この授業は難しい」などと勝手に決めつけてしまう者が多い。

この対策として今年度は、前回欠席した者は毎回チェックしておき（正当な理由のある欠席も含め）居残らせて、未履習の回のＡＶ講義録（授業内容を記録したビデオテープ。[1]-[5]参照）を見させて必ず補わせている。これにより「この授業は段階を踏めば理解できる、反面そうしなければたちまち理解できなくなる」ことが、大多数の学生に理解されるようになってきたように思う。（一度も出席しないなど論外の者を除く。また、正当な理由なしに欠席を繰りかえす者は年度途中でも「不可決定」を本人に通告し実質上授業から放り出すという「ムチ」も一方では用いている。）

５．おわりに

もちろんＬＯＧＯを計算機入門教育に用いることは私のオリジナルではない。４年半前に始めたころを思い出してみれば、全くの既存の手法のまねでスタートしたのだった。訪問者にコースの説明をする際「まあシーモア・パパート先生のまねを大学でやっているにすぎないのですが…」などと言っていたのを覚えている。

しかしその後私なりの小さな工夫も積みかさねてゆくうち、最初はコピーだったものもしだいに私の中で「消化」されて、自分の工夫した部分と共に、私にとっては一体の「私のもの」となっている。そしてその中には、判然とは区別しがたいが、私の「オリジナル」の部分もかなりまざっているはず、と思う。だからあるいは、本報告中「自分の工夫のごとく書いているがこれは誰々のアイディアでどこでもやっていることではないか。けしからん。」という部分もあるかも知れないが、もちろん故意の詐称ではなく、上記のような事情からなのだと御理解いただきたい。

参考文献

[1] 和田「計算機教育用映像システムと「ＡＶ講義録」」ソニー第２回懸賞論文「２１世紀のコンピューター：夢を現実に(1989)」応募論文（特別賞）
[2] ｂｉｔ別冊「知のキャンパス」Part.II, 第４章 pp.163-168.「長野大学」の節
[3] Wada "A contact point of computers and television - from an experience of constructing a video system for computer-science education", Proc. 5th International Joint Workshop on Computer Communications (Jul, 1990), KISS(Korea)&IPSJ(Japan), Kyongju, Korea., pp.181-186.
[4] 和田「ＡＶ講義録の夕べ」、情報処理学会プログラミングシンポジウム・夏のシンポジウム「計算機教育」(1991.07)、pp.87-96.
[5] 和田「ＡＶ講義録の夕べ（ＵＮＩＸ編）」第１８回日本ＵＮＩＸユーザ会（ＪＵＳ）ＵＮＩＸシンポジウム論文集(1991.11)、pp.103-111.