TSNETスクリプト通信

TSC編集委員会

目次

巻頭言

jscripter

TSNETスクリプト通信創刊1周年記念号、第5号、バージョン2.1を送り出す。今回は疲れ気味。ギリギリ。ニュースは出せなかったが、次号では是非まとめよう。

Yささんには、AWK版のテスト駆動開発を見せていただいた。そういうものかと初めて知った。勉強になるなあ。

機械伯爵さんは、階乗計算博物館を企画いただき、TSNETメンバーにも多数ご参加いただいた。機械さんが参加メンバーのリストを記事の最後にまとめていただいたので、最後の編集委員会のメンバーには特に掲載しなかった。当然のことながら、著作権等は同等の取り扱いということで、ご了承ください。機械さんのビジュアルで博物館的なまとめになるほどこういう構想だったのかと納得。ご苦労様でした。私自身は、様々な言語が実際に使えるようになったことを再認識した企画であった。豊かな時代になったものだとこの経済的には低落した時代を不思議に思う。

機械さんの「やｎでレPython ～新約蛇神祭祀書～」第3回がしっかりと出てきたのにはびっくり。凄い創作力だ。

海鳥さんはお忙しそうだが、「よしおさんとロボ太」の連続物「ヨシオサン」4話を転載させていただくことにした。

私は、MVCフレームワークをいろいろと触ってみた。まだ、そのフレームワークとしての意義や価値を見出せるほどの場所には辿りついていない。単に分けるために書き方やシステムが複雑になっただけではないかとも思える。そんなことを思いながらも、どのようにしてメリットが生まれるのかを追求してみたいという気持ちもある。さて、どこに漕ぎ着くだろうか。

次号は8月を予定している。お楽しみに。

TDD のすすめ

written by Yさ

「テスト駆動開発(Test-Driven Developnment)」ってご存知ですか？

簡単に言うと、プログラマはコードを記述する前にテストを書き、その後、テストに合格するのに必要なコードだけを書くといった開発スタイルのことです。

テスト駆動開発では、追加すべき機能や作業項目を記した ToDo リストから、単体テストのコードを生成し、そこから作成すべきプログラム本体のコードが記述され、最後にリファクタリング(コードの整理)されてプログラムの構造が確定します。

詳細は Kent Beck 著「Test-Driven Developnment:By Example」を見てください。Java と Python のコードが半分くらいを占めています。

テスト駆動開発を、簡単な例でご紹介してみたいと思います。...awk で。

1.1 準備

階乗計算をする関数 fact() を作るとします。

XP(eXtreme Programming)では xUnit 等のテストの自動環境テスティング・フレームワークが利用されますが、ここでは単純なテスト実行結果が確認できる骨組みを作ってみます。

"関数を適当なテストケースで呼び出し、期待した結果かどうかを判定する"といったものです。

```
----- [factTest.awk]
# 階乗を返す
#    n : 階乗にする数
function fact(n){
  return 0.0;
}


# ///// 以下はテスト用 /////
function CLR_RESULT() {TestCnt_=ErrCnt_=0;
  print "吟じます！ テストケースを実行したときに～～～¥n";
}
function CNT_RESULT(sw) {++TestCnt_; if(sw) ++ErrCnt_;}
function CHK_RESULT() {
  if(ErrCnt_==0){
    printf("¥n 合計 %d 全てのテストにパスしてたら～～～¥n", TestCnt_);
    printf("  なんだか今日いけそうな気がする～！！  あるとおもいます！¥n");
  }else{
    printf("¥n 合計 %d のテスト中 %d に失敗してたら～～～¥n", TestCnt_, ErrCnt_);
    printf("  そこには何らかのエラーが～～  あるとおもいます！¥n");
  }
```

```
}
function eval(fname,val,stat,test,  err){
  err=0;
  if(stat=="==" && !(val==test)) err=1;
  if(stat=="!=" && !(val!=test)) err=1;

  if(err) print "err: " fname "=[" val "]",stat,test;

  CNT_RESULT(err);
}


BEGIN{
  CLR_RESULT();

  eval("#00: n=0 ", fact("0"),"==",1.0);

  CHK_RESULT();
}
-----
```

実行すると、

```
=====
D:¥TDD>gawk -f facttest.awk

err: #00: n=0 =[0] == 1

合計 1 のテスト中 1 に失敗
=====
```

期待値が"1"なのに"0"だったというメッセージが出ています。(詳細は後述します)

とりあえず関数cnvJNum()を以下のようにしてみます。

```
-----
function fact(n){
  return 1.0;
}
-----
```

```
=====
D:¥TDD>gawk -f facttest.awk


合計 1 全てのテストにパス
=====
```

テストは問題なく終了します。

ではテスト駆動開発(TDD)で階乗計算をする関数 fact() を実装してみましょう。

1.2 関数 fact() の実装

TDD ではまず最初にテストケースを追加します。

```
-----
  eval("#01: n=1 ", fact(1),"==",1);
  eval("#01: n=2 ", fact(2),"==",2);
-----
```

省略しますが、テストを実行し、失敗することを確認します。

TDD ではテストが失敗している状態を Red と呼びます。例えば、Java 用の xUnit である JUnit の GUI では赤いバーが表示されます。

最もシンプルな、テストをパスしそうな実装をしてみます。

```
-----
function fact(n){
  if(n==2) return 2;
  return 1;
}
-----
```

(省略しますが)テストは成功します。

TDD ではテストが成功している状態を Green と呼びます。JUnit なら緑のバーが表示されます。

ちなみにこれは TDD で Fake It と呼ぶ定石です。Fake の後は、別の視点からのテストケースを追加し、テストを失敗させ、次に成功するようにロジックを追加します。

```
-----
 :
  eval("#02: n=3 ", fact(3),"==",6);
-----
```

テストをパスしそうな実装をしてみます。

```
-----
function fact(n){
  if(n==3) return 6;
  if(n==2) return 2;
  return 1;
}
-----
```

TDDではコードに重複があるならリファクタリングの余地が無いか見直します。if文とreturnが繰り返され重複しているのに気がつきます。それを解消します。

```
-----  [リファクタリング途中、計算式にn導入]
function fact(n){
  if(n==3) return n * (2 * 1);
  if(n==2) return n * 1;
  return 1;
}
-----
 ↓
-----  [リファクタリング途中、計算式にfact()導入]
function fact(n){
  if(n==3) return n * fact(2);
  if(n==2) return n * fact(1);
  return 1;
}
-----
 ↓
-----  [if,returnの重複除去、他]
function fact(n){
  if(n>1) return n * fact(n-1);
  return 1;
}
-----
```

省略しますがリファクタリング途中で、テストが失敗しないか常に確認します。全てのテストに成功する事で、まずい事が起きていないのが確認できます。

なお"機能の拡張"と"リファクタリング"は異なる概念なので、きちんと区別して考える必要があります。TDDでのリファクタリングは重複の除去についてのみです。(蛇足:作り直しは"リストラクチャリング"です。)

では、4以上のテストケースを追加してみましょう。

```
-----
 :
  fn=fact(3);
  for(n=4; n<=10; ++n){
    msg=sprintf("#02: n=%d ", n);
    fn*=n;
    eval(msg, fact(n),"==",fn);
  }
-----
```

(省略しますが)そのままでテストは成功しました。

そんな時もあります。

さらに大きな数も扱えるのかの確認は必要そうですが、この辺で終わりにします。

1.3 まとめ

今までのステップをまとめると次のようになります。

1) Red
テストを書いて失敗させる。
2) Green
テストを成功させる。Fake It も許される。
3) Refactor
テストが成功する状態を保ちながら、リファクタリングを進める。

こう言ってもいいでしょう。

1) Fail It
2) Fake It
3) Make It
4) Refactor It

プログラムが間違いなくプログラマの意図するとおりに動作しているか、執拗に確認しながら進んでいく TDD のやり方が分かっていただけましたでしょうか？

参考 URL：
例で覗くテスト駆動開発(TDD)
http://www.diana.dti.ne.jp/~katta/tdd/TddSamples.html

(2009 年 4 月 29 日)

階乗計算博物館

序

階乗計算博物館へようこそ。
様々なプログラム言語で綴られた、階乗計算手続き／定義／関数の数々。
皆さんのおなじみの言語、あるいは噂には聞くもののご覧になるのは初めての言語。
何かを感じて戴ければ幸いです。

---CONTENTS---

この博物館、まだまだ展示物が足りません。
ご覧の皆様の中に、珍しい言語、あるいは書き方をご存じの方、御寄贈戴けたら、と思います
（山上の垂訓の後のパンの如く、アイディアは共有しても増えはすれ減りません）

＜凡例＞
プログラミングに使用されている記号は、凡そ以下の通り
n,N …… 階乗計算する数（n!）
直値の場合は10（たまに違う場合もある）
fn, Fn …… n! の答え
fact …… 階乗関数名
その他 …… カウンタや内部関数など(x,y,fxなど)
基本的に、コンソールに入力されている形式の場合は、表示するコマンド／手続きを書きましたが、中には引数に数値を与えるだけで求められる関数の形でのみ記述されているものもあります。

第1室『AWK』

■ループによる手続きを関数化したもの
＜forループを使用した例＞
```
function fact01(n,  fn){
  for(fn=1; n>1; --n) fn*=n;
  return fn;}
```

＜再帰関数を使った例＞
```
function fact02(n){
  if(n>1) return n * fact02(n-1);
  return 1;}
```

（出品：Yさ）

■ワンライナ
＜通常のループ＞
```
C:¥prog¥gawk>gawk "END{n=10;fn=1;while(n>1)fn*=n--;print fn}"
^Z
3628800
```

＜半自動＞
```
C:¥prog¥gawk>gawk "BEGIN{fn=1}{for(n=1;n<NF+1;n++)fn*=$n}END{print fn}"
1 2 3 4 5
6 7 8 9 10
^Z
3628800
```

＜連想配列を用いた方法＞
```
C:¥prog¥gawk>gawk "END{fn[1]=1;n=10;for(i=2;i<=n;i++)fn[i]=fn[i-1]*i;print fn[n]}"
^Z
3628800
```

（出品：機械伯爵）

■ワンライナ（BEGIN部、計算途中表示、外部パラメータによる変数定義）
＜最終結果を表示しない＞
```
D:¥>mawk32 -v n=10 "BEGIN{ print fn=n; while(--n>1) print fn*=n; }"
10
90
＜中略＞
1814400
3628800
```

＜最終結果を表示する＞
```
D:¥>mawk32 -v n=10 "BEGIN{ print val[1]=1; for(i=2; i<=n; ++i) print val[i]=i*val[i-1]; }"
1
2
＜中略＞
362880
3628800
```

（出品：Yさ）

第２室『Erlang』

■再帰定義による推論

```
-----^ sample.erl
-module(sample).

-export([factorial/1]).

factorial(0) -> 1;
factorial(N) -> N * factorial(N-1).
-----$
```

```
-----^ 実行結果
Eshell V5.6.1  (abort with ^G)
1> c(sample).
{ok,sample}
2> sample:factorial(10).
3628800
3>
-----$
```

（出品：藤岡 和夫）

第3室『FORTH』

■変数を変化させる例

```
variable fn  ok
: fact 1 fn ! 11 1 ?do fn @ i * fn ! loop fn @ . ;  ok
fact 3628800  ok
```

■スタック操作のみ

```
: fact begin swap over * swap 1- dup 1 = until drop . ;  ok
10 9 fact 3628800  ok
```

■再帰

＜gforthのマニュアルのサンプル＞

```
: fact dup 0> if dup 1- recurse * else drop 1 endif ;  ok
10 fact . 3628800  ok
```

(出典：http://www.complang.tuwien.ac.at/forth/gforth/gforth-0.7.0.pdf)

＜改造版＞

```
: fact dup 1 > if dup 1- recurse * endif ;  ok
10 fact . 3628800  ok
```

＜スタックをあまり深くしないバージョン＞

```
: fact 1- dup 1 > if swap over * swap recurse else drop endif ;  ok
10 10 fact . 3628800  ok
```

※確認：gforth 0.7.0

（出品：機械伯爵）

第４室『Haskell』

■再帰定義

```
fac 0 = 1
fac n = n * fac (n-1)

Prelude> :load factorial
*Main> fac 10
3628800
```

（出品：藤岡和夫）

■演算子の連結

```
>ghci
Prelude> foldl (¥x y -> x * y) 1 $ take 10 [1..]
3628800
Prelude> foldl (*) 1 $ take 10 [1..]
3628800
```

■関数定義（再帰）

```
ffact :: Int -> Int
ffact n
    | n == 0  = 1
    | n > 0   = foldl (*) 1 $ take n [1..]
    | otherwise = error "invalid value"

Prelude> :load ffact
*Main> ffact 0
1
*Main> ffact 1
1
*Main> ffact 10
3628800
```

（出品：Bruce.）

第５室『JavaScript』

■再帰定義をブラウザで表示

```
<html>
  <head>
    <meta name="content-type" content="text/html; utf-8">
    <title>階乗のデモ</title>
    <script type="text/javascript">
function run() {
  var num = parseInt( document.fact.input.value );
  if ( num > 0 ) {
    document.getElementById( 'answer' ).textContent = '=> ' + fact( num );
  } else {
    alert( '自然数を入れてね' );
  }
  return false;
}
function fact( num ) {
  if ( num > 1 ) {
    return num * fact( num - 1 );
  } else {
    return num;
  }
}
window.onload = function() {
  document.fact.onsubmit = run;
};
    </script>
  </head>
  <body>
    <h1>階乗のデモ</h1>

    <form name="fact"><div>
      <input type="text" size="4" name="input">
      <input type="submit" value="階乗!">
      <span id="answer"></span>
    </div></form>
  </body>
</html>
```

（出品：ねこ丸）

第6室『Lisp(Common Lisp)』

■演算子の連結による

＜lambdaによる直接計算＞

```
((lambda (m)
  ((lambda (f x)
    (if (null x)
      1
     (let ((r (car x)))
      (setq x (cdr x))
      (while x
        (setq r (funcall f r (car x))
            x (cdr x)))
      r))) #'* ((lambda (n)
                (let ((r nil))
                  (while (< 0 n)
                   (setq r (cons n r)
                         n (- n 1)))
                 r)) m))) 15)
1307674368000
```

（出品：Bruce.）

＜関数定義を用いたもの＞

```
(defun fact (n Lf)
  (if (< n 2)
      (eval (cons '* Lf))
    (fact (- n 1) (cons n Lf))))
fact
(fact 10 ())
3628800
```

（出品：機械伯爵）

■再帰（参考）

＜通常＞

```
(defun fact (n) (if (= n 0) 1 (* n (fact (1- n)))))
```

＜末尾再帰＞

```
>(defun fact (n fn) (if (= n 0) fn (fact (1- n) (* fn n))))
>(fact 10 1)
3628800
```

（出典：『入門Common Lisp 関数型４つの特長とλ計算』毎日コミュニケーションズ 刊）

第7室『Lua』

■再帰

```
function factorial(n)
  if n == 0 then
    return 1
  else
    return n * factorial(n - 1)
  end
end
print (factorial(10))
```

（出品：藤岡 和夫）

■通常のループ

＜forループ＞

```
fn = 1
for n = 10, 1, -1 do
  fn = fn * n
end

print(fn)
```

＜while ループ＞

```
do
  local n, fn = 10, 1
  while n > 1 do
    fn = fn * n
    n = n - 1
  end
  print(fn)
end
```

＜repeat-untileループ＞

```
do
  n, fn = 10, 1
  repeat
    fn = fn * n
    n = n - 1
  until n < 2
  print(fn)
end
```

（出品：機械伯爵）

第8室『OCaml(Objective Caml)』

■再帰定義による推論

```
# let rec factorial = function
      | 0 -> 1
      | n -> n * factorial(n - 1);;
val factorial : int -> int = <fun>
# factorial 10;;
- : int = 3628800
#
```

（出品：藤岡 和夫）

第9室『Pascal』

■ループを用いた方法
＜repeat-untilループを使った例＞
```
program factorial(output);
var n, fn: integer;
begin
  n := 10; fn := 1;
  repeat
    fn := fn * n; n := n - 1;
  until n < 2;
  writeln(fn);
end.
```

＜goto文を使った原始的なループ＞
```
program factorial(output);
label 100;
var n, fn: integer;
begin
  n := 10; fn := 1;
  100:
    fn := fn * n; n := n - 1;
  if n > 1 then goto 100; writeln(fn);
end.
```

＜whileループを使った例＞
```
program factorial(output);
var n, fn: integer;
begin
  n := 10; fn := 1;
  while n > 1 do
    begin
      fn := fn * n; n := n - 1;
    end;
  writeln(fn);
end.
```

＜forループを使った例＞
```
program factorial(output);
var t, n: integer;
begin
  n := 1;
  for t:=1 to 10 do n := n * t;
  writeln(n);
end.
```

■再帰関数を使った例
＜単純再帰＞
```
program factorial(output);
function fact(n: integer):integer;
begin
  if n < 2
    then fact := 1
    else fact := fact(n - 1) * n
end;
begin
  writeln(fact(10));
end.
```

＜末尾再帰＞
```
program factorial(output);
function fact(n, t: integer):integer;
begin
  if n < 2
    then fact := t
    else fact := fact(n - 1, t * n)
end;
begin
  writeln(fact(10, 1));
end.
```

■型とリストを使った例
```
program factorial(output);
const
  n = 10;
type
  length = 1..n;
var
  s : array[length] of integer;
  c : length;
begin
  s[1] := 1;
  c := 1;
  repeat
    c := c + 1;
    s[c] := s[c - 1] * c;
  until c = n;
  writeln(s[n]);
end.
```

（出品：機械伯爵）

第10室『Perl』

■再帰

```
sub factorial {
    my $n = shift;
    if($n==0){
        return 1;
    }else{
        return $n * factorial($n-1);
    }
}
print factorial($ARGV[0]);
```

（出品：藤岡 和夫）

■演算子の連結による

＜文字連結と評価＞

```
$n=10;print eval join("*",(1..$n));<stdin>;
```

（出品：林　宏）

＜forループによるもの＞

```
for($n=10;$n>0;$n--){push(@an,$n);};print eval join('*',@an);
```

＜forループによるもの(0の階乗にも対応)＞

```
$fn=1;for($n=$ARGV[0];$n>0;$n--){$fn*=$n;};print $fn;
```

（出品：藤岡 和夫）

＜引数より得た数値を連結＞

```
perl6 -e '$n = shift @ARGS; say [*] (1.. $n)'
```

（出品：Bruce.）

■畳み込み関数を使用したもの

```
use List::Util qw(reduce);
print reduce { $a * $b } 1..10;
```

（出品：藤岡 和夫）

第11室『Prolog』

■再帰定義による推論

```
fact(0, 1).
fact(N, Fn) :-
    N > 0, N1 is N - 1, fact(N1, Fn1), Fn is N * Fn1.
```

(出典：http://www.geocities.jp/m_hiroi/prolog/prolog02.html)

＜再帰定義を整理＞

```
fact(0, 1).
fact(N, Fn):-
  fact(N1, Fn1),
  N is N1 + 1,
  Fn is Fn1 * N.
```

＜関数子を前置＞

```
fact(0, 1).
fact(N, Fn):-
  fact(N1, Fn1),
  is(N, +(N1, 1)),
  is(Fn, *(Fn1, N)).
```

（出品：機械伯爵）

第12室『Python』

■ループを使った方法
＜forループを使用した例＞
```
n, fn = 10, 1
for x in range(1, n+1): fn *= x
print(fn)
```

＜forループの改良例（外部変数の後処理）＞
```
n, fn = 10, 1
for x in range(1, n+1): fn *= x
else: del x
print(fn)
```

＜クラス定義によるブロックの実装(裏技)＞
```
class BLK:
  n, fn = 10, 1
  for x in range(1, n+1): fn *= x
  print(fn)
del BLK
```

＜whileループを使用した例＞
```
n, fn = 10, 1
while n: fn *= n; n -= 1
print(fn)
```

＜whileループの改良例(nは不変)＞
```
n = 10
fn = n, 1
while fn[0]: fn = fn[0] - 1, fn[1] * fn[0]
else: fn = fn[1]
print(fn)
```

■再帰関数定義
＜再帰関数を使った例＞
```
def fact(n):
  if n < 2:
    return n
  else:
    return fact(n-1) * n
print(fact(10))
```

＜lambdaで実装した再帰関数の例・その１＞
```
fact=lambda x: x if x < 2 else fact(x-1)*x
print(fact(10))
```

＜lambdaで実装した再帰関数の例・その２＞
```
fact=lambda x: x < 2 and x or fact(x-1)*x
print(fact(10))
```

＜関数名変更による誤動作を防ぐラッピング＞
```
def fact(n):
  fx=lambda x: x < 2 and x or fx(x-1)*x
  return fx(n)
print(fact(10))
```

＜末尾再帰・その１＞
```
def fact(x, e=1):
  if x < 2: return e
  else: return fact(x-1, e*x)
print(fact(10))
```

＜末尾再帰・その２＞
```
fact = lambda x, e=1: e if x < 2 else
fact(x-1, e*x)
print(fact(10))
```

■Yコンビネータを使った再帰関数の方法
＜Yコンビネータを使用した例・その１＞
```
Y = lambda f:(lambda x: lambda m: f(x(x))(m))(lambda x: lambda m: f(x(x))(m))
fn = Y(lambda f: lambda x: x if x < 2 else f(x -1) * x)(n)
print(fn)
```

＜Yコンビネータを使用した例・その２＞
```
Y = lambda f:(lambda x: lambda m: f(x(x))(m))(lambda x: lambda m: f(x(x))(m))
fn = Y(lambda f: lambda x: lambda t: t if x < 2 else f(x-1)(t * x))(n)(1)
print(fn)
```

＜Yコンビネータを使用した例・その３＞
```
n = 10
print((lambda f:(lambda x: lambda m: f(x(x))(m))(lambda x: lambda m: f(x(x))(m))
  )(lambda f: lambda x: lambda t: t if x < 2 else f(x-1)(t * x))(n)(1))
```

■リスト内包を使った方法
＜リスト内包を使った（悪用した）例＞
```
fn = [0 if p.__setitem__(0, p[0]*x) else p[0] for p in [[1]] for x in range(1, n+1)][-1]
```

＜リストではなく一時変数を無名クラスオブジェクトに置き換えた例＞
```
fn = [0 if setattr(p,'t',p.t*x)else p.t for p in[type('',(),{'t':1})()
]for x in range(1,n+1)][-1]
```

＜ジェネレータを使用してメモリを節約した例＞
```
for fn in (0 if setattr(p,'t', p.t * x) else p.t
  for p in [type('',(),{'t':1})()]
  for x in range(1, n+1)):pass
```

■リストアイテムの演算子による連結
＜文字列にして連結＞
```
def func(n):
  if n < 1: return 1
  else: return eval('*'.join(str(x) for x in range(1, n+1)))
```

＜文字列にして連結(lambda版)＞
```
func = lambda n: 1 if n < 1 else eval('*'.join(str(x) for x in range(1, n+1)))
```

＜畳み込み関数を使った方法＞
```
from functools import reduce
n = 10
print(reduce(lambda x,y:x*y,range(1,n+1)))
```

＜演算子関数を併用したバージョン＞
```
from functools import reduce
from operator import mul
n = 10
print(reduce(mul,range(1, n+1)))
```

■番外編
＜ライブラリを用いた方法＞
```
import math
print(math.factorial(10))
```

＜TkinterによるGUI＞
```
# fact.pyw
from tkinter import *
w = Tk()
l0 = Label(w, text='階乗計算')
e, l1 = Entry(w), Label(w, text='')
b = Button(w, text='計算',
command=lambda:l1.__setitem__('text',str((la
mbda n:[0 if p.__setitem__(0, p[0]*x) else
p[0] for p in [[1]] for x in range(1, n +
1)][-1])(int(e.get())))))
for x in l0,e,b,l1:x.pack()
w.mainloop()
```

（出品：機械伯爵）

第13室『R』

■順序列を生成し、全ての要素を乗算する

```
> x <- 10
> prod(seq(x))
[1] 3628800
```

（出品：Bruce.）

第14室『Ruby』

■畳み込みメソッドを使用
＜配列のinjectメソッドを使用＞
```
>irb
irb(main):001:0> [*1..5]
=> [1, 2, 3, 4, 5]
irb(main):002:0> [*1..5].inject(:*)
=> 120
```
（出品：Bruce.）

＜別解＞
```
(1..5).inject(:*)
```
（出品：まつもとゆきひろ）

第15室『Smalltalk』

■ループ構造

＜to:do:ループ＞

```
y := 1.
1 to: 10 do: [:x | y := x * y].
Transcript show: y. "do it"
```

＜whileTrue:を使ったループ＞

```
y := 1.
x := 10.
[x > 1]  whileTrue: [y := y * x. x := x -1].
Transcript show: y.
```

■畳み込みメソッドによる

```
n := 10.
oc := OrderedCollection new.
[|s| s := 1.
  n timesRepeat: [
    oc add: s.
    s := s + 1.
  ]
] value.
oc inject: 1 into: [:fn :n | fn * n].
```

■階乗計算、組み込みメソッド

```
10 factorial.
```

（出品：機械伯爵）

第16室『Tcl』

■ループ版
＜while版＞
```
% while {$x > 1} {set y [expr $y * $x]incr x -1}
% puts $y
```
（出品：機械伯爵）

＜for版＞
```
for {set n 10; set fn 1} {$n > 1} {incr n -1} {puts [set fn [expr $fn*$n]]}
```
（出品：JO3EMC）

■再帰版
＜通常の再帰＞
```
% proc fact {x} {if { $x > 1} { return [expr $x * [fact [expr $x - 1]]]} else { return 1}}
% fact 10
```
（出品：機械伯爵）

＜exprを減らした再帰関数＞
```
proc fact {n} {
  if {[incr n -1] < 1} then {return 1}
  return [expr ($n+1)*[fact $n]]
}
puts [fact 10]
```

■演算子の連結

＜for+join版＞
```
for {set n 10} {$n > 1} {incr n -1} {puts [lappend list $n]}
puts [expr [join $list *]]
```

■ライブラリを使用
```
package require math
puts [::math::factorial 10]
```
（出品：JO3EMC）

解説

■階乗計算と各プログラムの対応

ご存じの方も多いでしょうが、階乗とは「順々に掛ける」の意味で、1から順に掛ける計算です。

たとえば、5の階乗(5!と書きます)なら1×2×3×4×5で120、10の階乗なら1×2×3×4×5×6×7×8×9×10で3,628,800です。

見ての通り、直ぐに巨大な数値になってしまいます(10以降は1増える毎に確実に1桁以上増えますよね)ので、プログラミング言語の実装の中には、仕様の関係で、あまり大きな数値が扱えないものもあります。

よって全てのプログラムに於いて10!までは計算できることを確認するに止めました。

ちなみに、0!=1というルールがあるのですが、これについては、一部プログラムでは対応していません。

よって実質上、1!～10!に於いて正常に動くことが確認されていると思ってください。

なお、集められたコードの数については、単なる投稿数の差であり、言語の表現能力とは一切関係ありません。

■階乗計算のアルゴリズムについて

階乗計算をプログラムで解く場合、直接書き上げて計算する場合(1*2*3*4*5 など)を除き、大きく分けて３つの方法が考えられます

1.ループを使った方法

構造化以前のBASICなどでも可能な、手続き型プログラミングの基本的な方法です。

答えを収納する変数にカウンタとなる数値を掛けて計算します。

階乗計算の数値をＮとすると、

```
Ｎ→変数Ａ
１→変数Ｂ
ラベル
変数Ａ×変数Ｂ→変数Ｂ
変数Ａ－１　→変数Ａ
変数Ａ＞１ならばラベルへ
変数Ｂを表示
```

この方法では0!が計算できませんので、もし0!まで正確に表示するなら、ループの前に条件分岐を置く必要があるでしょう。

このアルゴリズムの変形として、ループテストを先頭に置く方法、及び１～Ｎの数値配列を製作して順々に掛けていく方法などがあります。

プログラミングを始めたばかりの初心者にも、比較的わかりやすい方法ですが、関数型プログラミングを推奨する人々からは、カウンタとなる変数への再代入が参照透過性を阻害する点に問題があると言われます。

2.リストの要素の演算子による連結

プログラミング言語の中には、順序数リストをリテラル（書式）として書けるものもあります。

このような機能がある場合は、リストの各要素を演算子（今回は乗算演算子）で連結することによって、いとも簡単に直接階乗の式そのものを生成してしまうことが出来る場合があります。

仕組みは非常に簡単なのですが、言語の機能に大きく依存する方法でもあります。

[1,2,3,4,5] ⇒ 1×2×3×4×5

関数型プログラミングでよく使用されるreduce関数の使用は、このタイプの変形であると言えるでしょう。

3.再帰的定義を用いた方法

階乗計算n!は、以下のように定義できます。

$$0! = 1$$
$$n! = (n - 1)! \times n$$

例えばn=5であれば、

$$\begin{aligned} 5! &= 4! \times 5 \\ &= (3! \times 4) \times 5 \\ &= ((2! \times 3) \times 4) \times 5 \\ &= (((1! \times 2) \times 3) \times 4) \times 5 \\ &= ((((0! \times 1) \times 2) \times 3) \times 4) \times 5 \\ \because\ 0! &= 1 \\ &= ((((1 \times 1) \times 2) \times 3) \times 4) \times 5 \\ &= 1 \times 1 \times 2 \times 3 \times 4 \times 5 \\ &= 120 \end{aligned}$$

となります。

自分自身を定義に含む定義を『再議的定義』と言います。

推論型プログラミング言語であれば、この定義をそのまま書くことで、あとは階乗計算のする整数さえ与えてやれば、勝手に計算してくれます。

手続き型プログラミング言語では、関数定義を用いて再帰的定義を実現します。

階乗関数：引数として変数Ａ
変数Ａ＝０
１を返す
それ以外
階乗関数（Ａ－１）×Ａを返す

再帰関数の変形として、末尾再帰型があります。

※Yコンビネータという特殊な式を使う場合も、基本的なアルゴリズムは再帰関数と同じです。

階乗計算は、もともと再帰的定義の例題として用いられることの多いテーマでもあります。

手続き型言語であっても、殆どが関数の再帰的定義を許していますが、実装の関係で呼び出しのネストが深くなることで、扱える数値の大きさとは別個の問題を引き起こす場合があります。

この問題に関して、末尾再帰であれば単純ループとして最適化する実装もあるのですが、全ての言語の実装がそのような最適化ができるわけではありません（意図的にそういった最適化をしないようにしている言語もあります）

また再帰定義は、慣れないと直感的とは言いがたいので、コードの可読性がやや低くなるかもしれません。

（コード編集・文責　機械伯爵）

【今回、ご参加戴いたTSNETの皆さん(sort順)】

Bruce. さん
JO3EMC さん
Ｙさ さん
ねこ丸 さん
まつもとゆきひろ さん
藤岡 和夫 さん
林　宏 さん

やｎでレPython
～新約蛇神祭祀書～

機械伯爵

Vikings:
(singing) Spam, spam, spam, spam, spam ... spam, spam, spam, spam ...
lovely spam, wonderful spam ...
Waitress:
Shut up. Shut up! Shut up! You can't have egg, bacon, spam and sausage without the spam.

ヴァイキング：
（歌う）すぱむ、すぱむ、すぱむ、すぱむ、すぱむ・・・すぱむ、すぱむ、すぱむ、すぱむ……かわいいすぱむ、すてきなすぱむ
ウェイトレス：
うるさいうるさいうるさいっ！　卵・ベーコン・スパム・ソーセージにスパム抜きなんてないのぉーっ！

『MONTY PYTHON'S FLYING CIRCUS』

■登場人物
・錦織真武（にしこり　まなぶ）：プログラミング入門者＆犠牲者
・羽生一子（はぶ　いちこ）　　：パイソニスタ＆電波系狂信者

第三回　止まらない未来を目指して

『くりかえしがカンジンなんだよ』
　電話口の向こうのイチコが笑ったような気がした。
　ケータイの通信記録を見ると、メールを合わせて一日に三桁を突破していた。
　噂によれば、女の子同士ならこんなことも珍しくないそうなのだが、そんなヘヴィなケータイ生活の経験が無いマナブにとっては、くすぐったいような嬉しさとともに、若干の戸惑いと疲労とそして不安が芽生えていた。
　そんなマナブがふと「よくわからないけど、僕らのメールのやりとり回数って多くないのかな？」と、漏らしてしまった言葉の返事が『くりかえし』だった。
『コトバでもプログラムでも、くりかえしがダイジなんだよ。コトバをカサねるゴトに、あたしはマナブくんにちかづいてるキがするよ』
「無限に近づく、漸近線？」
　あはは、とケータイの向こうでの笑い声。
『ウマいことイうね。そう、ココロはゼンキンセン。ムゲンにチカづいてもカサならない』
　そして、一拍後。
『だから、**ゼロにしてしまえばかさなる**んだよね』
　……
「コンピュータと違って、ゼロは『無い』んだけど……」
　ゼロは、計算上の便宜上の概念。
『あ、そうだね。ゼロはギだもんね。ごめん、ちょっとコンランしてたよ』

混乱して何を考えていたのか……
聞くのが怖かった。

「というわけで、Python教室、第三回目だよっ！」
例によってマナブの部屋。
イチコのハイテンションと裏腹に、マナブはやや疲れ気味だ。
「どうしたの、マナブくん？　なんか疲れてない？」
「気にしなくていいよ」
イチコとつき合う以上は、もうコレは慣れなければならないコトなのだろう。
「今回は『繰り返し』だよ」
「繰り返し？」
つい最近、聞いたような……
「そう、コンピュータの計算って、繰り返してこそ、だから。**繰り返さない計算機はただの計算機**だよ」
意味不明。
「例えば、階乗計算って知ってる？」
「知らない」
学校では習っても、常識にある言葉ではない筈だ。
「階乗っていうのは、1から、ある整数までの数値を、順番に掛けた数値だよ。例えば『5の階乗』なら、『1×2×3×4×5』で 120 だよ。ちなみに記号では'5!'みたいに、数値の後ろに ！ って書くよ」
「……何に使うの、それ？」
「組み合わせ計算によく使うよ。たとえば、10 個の中から 3 個選ぶ時の組み合わせは、10!÷(3!×(10 - 3)!)で計算できるんだよ」
数学は嫌いだ。
「で、それが何の役に立つの？」
自然と挑むような口調になる。
「例えば、連勝複式の競馬の組み合わせから、確率計算とか……」
「馬のデータ無視では勝てないと思うけど。それに、競馬とかやんないし」
確かに、未成年がやるものではない。
「カードゲームで、次に欲しい手が来る確率とか」
「う〜ん」
微妙。
「MAHAMANを使って、望みの効果が現れる確率とか」
「あ、**それは結構重要**かも」
……そうなのか？
「あと、RPGでモンスターのトレジャー確率とか……」
「**死活問題**だね」
マナブの死活問題って一体……。
「階乗計算する方法は色々あるんだけど、一番簡単なのはこんなのだよ。たとえば、10!を求めるなら……」

```
Fn = 1
n = 10
while n > 1:
  Fn = Fn + n
  n = n - 1
print(Fn)
```

「〈while〉って見たことないけど……書き方からすると、〈if〉みたいなの？」
　嬉しそうにうなずくイチコ。
「いい勘だね、優秀だよマナブくん」
「そんなのもなのかなぁ」
　なんかピンとこない、と思うマナブだったが、一方、イチコが嬉しいならいっか、とか思うのだった。
「マナブくんの言うとおり、〈while〉はifに似てるよ。〈while〉の横の式が条件で、その条件が真である限り、ブロックの中を繰り返し実行するんだよ？」
「'>'はそのまま、不等号の『大なり』でいいの？」
「うん。どうして？」
「ほら前回、等号がアレだったから」
「あ、そっか。うん、不等号は大なりも小なりもそのまま。他にも『≦(以下)』は'<='、『≧（以上）』は'>='、そして『≠（等しくない』は'!='でできるよ」
「ふぅん？　それって、止まるの？」
「止まるよ？」
「ブロック内の文を実行すると、条件が変わるの？」
「うん、ちょっとココ見て欲しいんだけど……」

```
n = n - 1
```

「……なにこれ？」
「えへへ。ちょっと変でしょ？」
「変も変だよ。n が n - 1 のわけ、ないじゃない」
「うん。でも、イコールって代入の記号だよね？」
「あ、そっか……でも、代入とかいっても、 n の値に n - 1 を入れちゃったら変じゃない？」
「どうして？」
「例えば、n が 10 だったとして、n - 1 なら 9 だけど、9 を入れた途端、n は 9 で、n - 1 は 8 になるわけで……」
　ふふふ、とイチコが意味深げに笑う。
「マナブくんの考え、ある意味では間違ってないよ」
「ある意味？」
「うん。プログラミング言語によっては、そういう扱いをするものもあるから。でも、Pythonは割とクラッシックな手続き型言語だから、『評価』のタイミングで、こういう書き方ができるんだよ」
「手続き型言語？」
「うん。作業の内容を書き記すタイプの言語、って言えばいいかな？」
「どういうこと？」
「具体的に、ここで何が起きているか、説明するね。まず代入文では、右辺が評価されるんだよ。評価っていうのは、ここではとりあえず『計算する』って思えばいいよ。例えば n が 10 だとしたら……

```
n - 1 ⇒ 9
```

となって、右辺が 9 に置き換わるんだよ。だから、

```
n = 9
```

ってなるわけ」
「それじゃぁやっぱり、いつまで経っても止まらないじゃない」

「どうして？」
「だって、ここに来る度に、n が 9 になるんでしょ？　9 は 1 より大きいから、止まらない」
「マナブくん？」
「はい？」
「今度ここに来る時は、n は 9 だから、n = n - 1 を通った時に、次は 8 になると思わない？」
「あ……」
　言われてみれば、そのとおりだった。
「うっわー、マヌケだ。さっきせっかく褒められたのに、これじゃダメじゃん」
　イチコはフルフルと首を振った。
「ううん、そうじゃないよ。マナブくんはまだ、プログラミングの考え方に慣れていないだけ。そこで間違えるのは別におかしくも恥ずかしくもないよ」
「そうなの？」
「うん。むしろ何の疑問も無く『これはこういうものなんだ』って覚えちゃうよりずっといいよ。例えば、今までの知識だけで書くなら、こんな風に書いた方がわかりやすかったかもしれない」

```
x = n - 1
n = x
```

「あ、もしかしてコレと同じなの？」
「n の結果は同じだけど、x っていう余分な変数が入るから、こういう書き方はしないんだよ」
「それで、よく使う書き方を？」
「ううん、そうじゃなくって、マナブくんには評価のタイミングについて知ってほしかったから、わざとこういう書き方したんだよ。プログラミングを続けるなら、いずれ知らないと書けないことだしね」
「右辺を評価してから、その結果が左辺に代入される、ってこと？」
「うん、もう判ったでしょ？」
「ばっちり。結局、さっきイチコさんが言った『作業』って言葉で言えば、n = n - 1 は、『 n に 1 を足す作業』と思えばいいのかな？」
「うん、ホントにばっちりだね。じゃついでに、もう少し省略した書き方を見てみようか」

```
n -= 1
```

「これは、n = n - 1 と全く同じ。でも、これを最初に書くと'-='っていう特殊記号の使い方だけで終わっちゃうよね。でも、右辺に左辺のターゲットとなる変数を含んだもっと複雑な式は、全部こんな風に書けるわけじゃないから、まずは評価のタイミングを知っておくことが大切なんだよ」
　どうやらイチコは、相当に考えてコードを書いているらしい。
　今更ながらその熱意には頭が下がる。
「それじゃぁ、fn = fn * n も、fn に『その時の n 』を掛けていくってこと？」
「うんうん」
「とすると、最初 10 から始まって……あ、そっか、だから階乗になるんだ」
「そのとおりぃ！　掛け算は交換法則が成り立つからね」
「なるほど。あ、だとすると、n > 1 ってコトは 2 までだけど、1 は別に掛けなくてもいいからだね？」
「そうそう」
「最初に fn に 1 を入れておいたのも、掛け算しかしないんだったら、1 だったら変化しないからだね？」
　マナブとしては自身があったのだが、最後の最後で、イチコは苦笑するような顔を見せた。
「うん……そうなんだけど……、ホントは最初に n に 0 を入れても成り立つように、なんだよ」
「え？」

「これは知らなくても当然なコトなんだけど、0! = 1 っていうお約束があって、それが成り立つようにしてあるんだよ」
「あ？　え？　な、なんで？　1 からならわかるけど、0 からだと意味がわかんないよ。それに、n に最初に 10 って入れてあるんだから、それは変わらないわけだし……」
　すると今度は、挑むようにニヤリと笑うイチコ。
「一つずつ答えるね。まず、0! が 1 っていうお約束だけど、これが成り立つと、n 個から r 個のモノを選び出す公式が、少し幅が広くなるんだよ」

$$n! \div (r! \times (n - r)!)$$

「これがさっき出した式の一般項だよね。このとき、もし n = r つまり、n 個から n 個を選び出すとなれば、当然 1 通りとなる筈だけど、n − r が 0 になっちゃったら、0! を分母に持つことになるよね。もし 0! が 0 だと、分母が 0 になるから、計算できなくなるよね？　n より沢山選び出せたら手品だけど、現実にできることが計算で出来ないと変だよね。だから便宜上、0! を1 として計算して、つじつまを合わせてるんだよ」
「へぇ、そうなんだ」
「それからもう一つの答えだけど、前にやった input 関数、忘れてない？」
「え？　あ？　ああーっ！」
　イチコはニヤニヤ笑いながら、ディスプレイを指差す。
「はい、では復習。入力した数値の階乗を計算するプログラムを書きなさい」
「エス！　マム」
「あ、今回は例外処理は省略していいからね」

```
n = eval(input('整数の入力して下さい：'))
fn = 1
while n > 1:
  fn *= n
  n -= 1
print(fn)
input()
```

「はい、よくできました」
　……
「あ、あの、もしかしたら凄くバカな質問かもしれないけど……

$$1 * 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10$$

って書いた方が速くない？」
「 10! だとそうかもね。でも、100! とか 1000! とか、書く？」
「で、できるの？」
「せっかく書いたんだから、やってみたら？」
「うん」

```
整数を入力してください：100
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

「うわ」

```
整数の入力して下さい：1000
4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084
8696940480047998861019719605863166687299480855890132382966994459099742450408707375991882362
7727188732519779505950995276120874975462497043601418278094646496291056393887437886487337119
1810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534
2864657661166779739666882029120737914385371958824980812686783837455973174613608537953452422
1586593201928090878297308431392844403281231558611036976801357304216168747609675871348312025
4785893207671691324484262361314125087802080002616831510273418279777047846358681701643650241
5369139828126481021309276124489635992870511496497541990934222156683257208082133318611681155
3615836546984046708975602900950537616475847728421889679646244945160765353408198901385442487
9849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351
4733161170210396817592151090778801939317811419454525722386554146106289218796022383897147608
8506276862967146674697562911234082439208160153780889893964518263243671616762179168909779911
9037540312746222899880051954444142820121873617459926429565817466283029555702990243241531816
1721046583203678690611726015878352075151628422554026517048330422614397428693306169089796848
2590125458327168226458066526769958652682272807075781391858178889652208164348344825993266043
3676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063
4463179746059468257310379008402443243846565724501440282188525247093519062092902313649327349
7565513958720559654228749774011413346962715422845862377387538230483865688976461927383814900
1407673104466402598994902222217659043399018860185665264850617997023561938970178600408118897
2991831102117122984590164192106888438712185564612496079872290851929681937238864261483965738
2291123125024186649353143970137428531926649875337218940694281434118520158014123344828015051
3996942901534830776445690990731524332782882698646027898643211390835062170950025973898635542
7719674282224875758676575234422020757363056949882508796892816275384886339690995982628095612
1450994871701244516461260379029309120889086942028510640182154399457156805941872748998094254
7421735824010636774045957417851608292301353580818400969963725242305608559037006242712434169
0900415369010593398383577793941097002775347200000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000
```

「う、うゎうゎうゎぁ……」
「電卓じゃ計算できないって、わかった？」
「うん。モノスゴ。びっくり。あ、そうか、だから『繰り返さない計算機はただの計算機』なんだ」
「そういうこと」
「っていうか、自分のパソコンでこんなコトができることに、びっくりしてるんだけど……」
「人間の眼に見えないトコで、もっとスゴい計算、してるんだよ」
「そうだったんだ」
「それもその筈だよね。アポロ計画に使ったコンピュータの能力って、ファミコンのチップと同じくらい、って聞いたことあるから」
「スーファミ？」
「オリジナルのファミリーコンピュータ」
マナブの脳裏に、FM 音源の電子音を鳴り響かせながら月に向うロケットが浮かんだ。
「ところでマナブくん、今、プログラムを２回起動させたよね？」
「うん」
「でも、使うプログラムが一つなんだから、起動一回で、続けて使えたら便利だと思わない？」
「そ、そうだよね。面倒だった」
「ついでに、何回でも使えて、止めたい時に止められると便利だと思わない？」
「まぁ、ソレは……」

「できるよ」
「え？」
「できるんだけど、マナブくん、自分で書いてみない？」
「え？　え？　えええええーっ！」
　頭が真っ白になった。
「ふふふ、ちゃんとヒントはあげるよ。ヒントその１、今日やったループを使うこと。ヒントその２、終了条件は、特定の数値の入力を監視すると簡単だよ」
「あ、あ、あーーー、なんか、なんか……こう……」
「ヒントその３、ループを二重にするといいよね……」
「トライします教官」
「ヒント、もういい？」
「とりあえずは」
　そして、画面の上で、あーでもないこーでもないとひねくりまわし、時には暴走させ……
　下のコードが出来上がるまで、結局30分ほどかかった。

```
n = 1
while n != 0:
  n = eval(input('整数の入力して下さい：'))
  fn = 1
  while n > 1:
    fn *= n
    n -= 1
  print(fn)
  input()
```

＜実行結果＞

```
整数の入力して下さい：5
120

整数の入力して下さい：10
3628800

整数の入力して下さい：15
1307674368000

整数の入力して下さい：20
2432902008176640000

整数の入力して下さい：0
1
```

「うんうん、上出来上出来、だよ」
「でもイチコさんは、この後、手直しするんだよね？」
「えへへ、わかった？」
「毎度のこと、だから」
「あ、でも、ここまで教えた内容では、ホント、パーフェクトだよ」
「では、更なる高みをめざして」

「うんうん、優秀な生徒で嬉しいよ」

```
# fact2.py
n = 1
print('階乗計算を行います')
print("終了するとき'0'を入力してください")
while 1:
  ni = eval(input('¥n整数の入力して下さい：'))
  if ni == 0:
    break
  n = ni
  fn = 1
  while n > 1:
    fn *= n
    n -= 1
  print(ni, 'の階乗は', fn, 'です')
  input('¥n(Enter キーを押して次に進んで下さい)')

print("'0'が入力されましたので終了します")
input('Enter キーを押して終了して下さい')
```

＜実行結果＞

```
階乗計算を行います
終了するとき'0'を入力してください

整数の入力して下さい：5
5の階乗は120です

(Enter キーを押して次に進んで下さい)

整数の入力して下さい：10
10の階乗は3628800です

(Enter キーを押して次に進んで下さい)

整数の入力して下さい：0
'0'が入力されましたので終了します
Enter キーを押して終了して下さい
```

「コード、かなり意味不明だね」
「わかんないトコ聞いて。説明するから」
「じゃ、まずどしょっぱなから。# fact2.py って何？」
「'#'はコメント記号だよ。'#'記号から改行するまで、コメントとして内容はプログラムから無視されるんだよ。今回みたいに先頭から書いてもいいけど、行の途中から書いても問題ないよ」
「え？　実行されないもの、なぜ書くの？」
「メモだよ。例えば人にプログラム渡したとき、その内容が分かりやすいように、とかね。でもホントは自分のために書くことが多いんだよ。Python はコードの読み易さ……可読性っていうんだけど、これを最重要視した言語ではあるんだけど、それでもやっぱり、何のコメントも無いと読むのが大変なコードは当然あるんだよ。だから、こういうコメントが書けるようになってるんだよ」

「今回は何を書いたの？」
「ん、あんまり意味ないけど、こういうファイル名にしてね、って感じ」
「ふうん」
「良いコメントは良いプログラムの必須条件だから、おいおいコメントのつけ方も教えてあげるね」
「うん」
「あとは？」
「じゃ、次は〈while〉の条件。1 っていくらなんでも変だよ」
「そうだね。これはちょっと難しいコトだよ。'=='や'>'のような記号を『比較演算子』って言うんだけど、このタイプの演算子は、『真偽値』を返すんだよ」
「真偽値？　返す？」
「『返す』っていうのは『評価されて入れ替わる』って考えればいいよ。例えば『 2 + 3 は 5 を返す』みたいな感じ。ＯＫ？」
「ヱス、マム」
「『真偽値』は文字どおり『真』か『偽』の値しかもたない値だよ。真は'True'、偽は'False'で表されるよ」
「あ、ああ、なんか数学でやった覚えがある。命題が真か偽か、とかなんとか」
「うん。そういうこと。例えば 5 > 3 は真だけど、5 < 3 は偽だよね」
「つまり、今までやってた『条件に合うか合わないか？』って、『真か偽か？』のどちらかって見てただけ、ってこと？」
「ご明察。そしてPythonでは、数字の0は偽、0以外は真と判定されるんだよ」
「じゃぁ、別に 1 でなくとも？」
「通るよ。ただ、1は反応が早いから、そう書いただけ。分かりやすさを考えるなら、

```
while True:
```

と書くのがいいかもね」
「じゃぁ、次の質問。〈while〉の条件が常に真だとすると、ループって止まらないんじゃない？」
「止まらないよ」
「え？」
　イチコはいたずらっぽく笑う。
「止まらないんだよ。だからこの書き方を『無限ループ』って言うんだよ」
「じゃぁ、なぜ実際に動かしたら止まるの？」
「なぜだと思う？」
　マナブは画面のコードをじっとにらみつけた。
「コレかっ！」
　指した指の先には〈break〉の文字。
「ナイス判断」
「正解？　ということは、〈if〉ブロックの中だから、〈if〉の条件が合えば、ループがブレイク（破れる）ってこと」
「そうそう」
「……なんか変だなぁ」
「え？」
　マナブの反応に、意表をつかれたイチコが抜けた声を上げる。
　マナブは気づかずに、画面を指す。
「ここだよ。〈break〉は〈if〉のブロックの中にあるんだよね。〈while〉のブロックの内側の〈if〉のブロックの命令が、外側の〈while〉の命令を破るって、なんかおかしくない？」
「……」
「え？　あれ？　僕、なんか変なこと言った？」
　呆けた顔から、少し頬を上気させながら、イチコは首を振った。

「ううん、びっくりした。**ソレって、普通、気づかないトコ**だよ」
「えっと、もしかしてイチコさんも知らないこと？」
「ううん、知ってる。知ってるけど、それは色々な言語を知って、比較して、Pythonの特徴を知ってるから判るだけだよ。比較なしにそんなこと、普通考えないもん」
「えっと……」
「マナブくんが今指摘したのは、プログラムの**構造に関する考え**方なんだよ」
「そうなの？」
「うん。結論から言えば、〈if〉や〈while〉のブロックは『制御ブロック』って言って、プログラムの実行の区切りを表すだけのものなんだよ。〈if〉は〈break〉を発動する条件であって、実行されてしまえば、ほかの〈while〉の中の実行文と何の違いも無いんだよ」
「あ、そうなの」
「でも『実行の区切り』じゃなくって『実行単位』のブロックも、実はあるんだよ。『定義ブロック』って言ってね、これは言わば『世界を区切る』ブロックなんだよ。この中での処理は、特別に外に影響を与えることを指定しない限り、外には影響を与えない隔絶された環境になるんだよ」
　キツネにつままれたような顔をしているマナブに、イチコは優しく微笑む。
「定義ブロックのことは、この次にするから、今回は考えなくていいよ」
「うん、そうする」
「じゃ、次は？」
「あとは大体わかるかな」
「そう」
「あ、でも、疑問じゃないけど質問。例えばこれ、〈while〉のループが二重になってるよね」
「うん」
「もし、二重のループを一気に抜けたい場合ってどうするの？　たとえば、結果が百万を越えたら計算をやめる、とか」
「マナブくん」
　ふとイチコを見ると、俯いてぷるぷると震えている。
（や、やばっ！　な、なんか僕、地雷踏んだ？）
「あ、あのイチコさん……！！！」
　おずおずと呼びかけた瞬間……

　イチコが抱きついてきた。

「え？　え？　ええ――っ！？」
「……マナブくん、**スゴい、スゴいよ**」
　潤んだ目で見上げられて、こんなせりふを吐かれたマナブの頭は、一気に沸騰する。
　密着したところから、イチコの体温が、実温度以上の体感を持って、まるで燃えるように激しく感じられる。
　イチコは猫のようにマナブの胸に頭をすりつけると、すっっと身を引いて、キーボードを叩いた。

```
# fact3.py
n = 1
print('階乗計算を行います')
print("終了するとき'0'を入力してください")
try:
  while 1:
    ni = eval(input('¥n整数の入力して下さい：'))
    if ni == 0:
      print("'0'が入力されました")
```

```
      break
    n = ni
    fn = 1
    while n > 1:
      fn *= n
      if fn > 1000000:
        raise Exception('1,000,000 を超えましたので、停止／終了します')
      n -= 1
    print(ni, 'の階乗は', fn, 'です')
    input('¥n(Enter キーを押して次に進んで下さい)')
except Exception as e:
  print(e)
else:
  print("'終了します")

input('Enter キーを押して終了して下さい')
```

＜実行結果＞

```
階乗計算を行います
終了するとき'0'を入力してください

整数の入力して下さい：5
5 の階乗は 120 です

(Enter キーを押して次に進んで下さい)

整数の入力して下さい：10
1,000,000 を超えましたので、停止／終了します
Enter キーを押して終了して下さい
```

「えっと、前やった『例外処理』だっけ？」
「うん、よく覚えていたね」
　イチコに誉めてもらおうと思って必死で復習したのか、イチコの『スイッチ』が入らないように死に物狂いで復習したのか、については言わぬが花。
「今回は、例外をキャッチするんじゃなくて、わざと発生させてるんだけどね」
「もしか……しなくても、二重ループを抜けるため？」
「そうだよ。他のプログラミング言語では、二重ループを抜けるためだけに、GOTO 文を入れたり、ループにラベルを貼ったりして break を拡張したりするけど、Python では例外処理を利用するのが普通」
「GOTO 文って？」
「ラベルを貼ってある場所に自由に制御が移せる命令。Python には無いよ」
「ああ、スパゲッティ」
「そうそう」
「ラベルを貼るっていうのは？」
「ラベルは、プログラムコードの中に書くしるしで、GOTO や条件つき break は、そこにジャンプするんだよ」
「で、Python はなぜ例外処理を使うの？」

「例外処理を使用すると、処理の単位がどこまでか、がはっきりわかるからだよ。ちょっとコードは長くなっちゃうのが難点だけどね」
「あと、Exceptionにカッコが付いて、何か書いてる」
「他のエラーじゃなくって、こちらが指定したエラーだよっ、って判るように書いたんだよ」
「他のエラーと混乱しないの？」
「ホントは、オリジナルの例外を作る方法があって、そこでその例外処理のためのexcept節を書くのが本式なんだけど、ちょっと話があまりにも難しくなるから、今回はやめたんだよ」
「……今回、このくらいにしない？」
「うん。ホントはもう一つのループの話もしたかったけど、マナブくんの脱線があまりにも素晴らしすぎて、余計な話しちゃったしね。今日はもう十分、でいいよ」
「〈while〉以外にも、ループってあるの？」
イチコが頷く。
「〈for〉ループっていうのがあるよ。〈while〉ループは、条件から外れるまでループするけど、〈for〉ループはある特定の集まり全部に同じ処理を行う、っていうような時に使うんだよ。でも、よく考えたらそのために、まず『集まり』の話をしないといけないから、結局もうちょっと先だね」
「Pythonで習うこと、まだたくさんあるの？」
「いっぱいあるよ。でも、次に予定してる『関数』の話を聞いたら、大体のプログラムなら書けるようになる筈だよ。あ、そうそう、それとオブジェクトの話をちょっとだけ、ね」
「いっぱいかぁ、じゃ、とうぶんコレ、終わらないね？」
「マナブくん……」
「ん？」
「**早く終わってほしい？**」
今度こそ本当にスイッチが入ってしまったらしい。
イチコが、持ってきたポーチの中に手を入れた瞬間、マナブはイチコの両肩に手を書ける。
「さ、最後まで、**ずっとずっと、教えてくれるんだよね、イチコさんっ！**」
「……う、うん。がんばろーね」
イチコは、顔を赤くしながら、ポーチの中でつかみかけたものを離した。
それが何だったのか、マナブは知りたくもなかった。

【内容】
注意：本文および注釈の全てに於いて、Pythonはver.3.0以降を指します。

第三回目：
・while ループ
・比較演算子 >, <, >=, <=, !=
・評価の順序
・演算代入子 -=, *=
・コメント
・無限ループ
・break文
・真偽値(boolean数)True, False
・raise 文
・引数つきのException

【注釈】もう、誰が対象なんだかよくわかりません

・スパムスケッチ
MONTY PYTHONの中でも超有名なアレ。ちなみに、知らない人のために蛇足ながら言っておくと、訳はなんか可愛くなってしまったが、ウェイトレスはテリー・ジョーンズ（♂）です。

・止まらない未来を目指して
知っていても絶対に自慢にならない歌のタイトルパート２。とはいっても、前回よりはかなり有名かも。

・ゼロは『無い』
数学上の概念が、全て現実世界に適用できるわけではない。の、筈なのだが、どーも数字の呪いや黒魔術に翻弄されるのが人間の性(さが)らしい。

・ゼロはギ
『偽』のこと。詳しくは本文参照。

・学校では習っても、常識にある言葉ではない筈だ
もし、学校で習うことが常識ならば、この世は教養で満ちあふれている筈だ（そんな薄ら寒い世界は勘弁して欲しい）

・10 個の中から 3 個選ぶ時の組み合わせは、10!÷(3!×(10 - 3)!)で計算できる

順列とか組み合わせとか嫌いだーっ！　とか言う人のために、無理やり数学のおさらい（毒笑）

※知ってる人は読み飛ばして下さい。

＜順列・組合せ＞
順列は、並べ方の種類が何通りあるか、という話。
たとえば1～5のナンバーのついた5枚のカードを一列に並べるとすると、最初の1枚目に来るカードの可能性は当然5種類ある。
で、その5種類の夫々(それぞれ)に対して、2番目に選べるカードは4種類あることになる。
つまり、先頭が1のカードなら、2番目に置けるカードは2,3,4,5の4枚。

先頭が2のカードなら、2番目に置けるカードは1,3,4,5の4枚となる。
同様に、順番が進む毎に選べるカードの種類は減っていき、最後の5枚目については、4枚が選ばれてしまったら選ぶ余地が無い。
1番目が5種類、その夫々に2番目が4種類、その夫々に3番目が3種類、その夫々に4番目が2種類、言わなくてもいいけど、5番目は1種類。
つまり、5の順列は5×4×3×2×1、つまり5!。
そう、階乗計算をすると、順列が求められるというわけである(nの順列はn!)。
さて、5枚から3枚選んで並べる順列は、この順列の先頭の3つ、すなわち5×4×3になる。
これは、5×4×3×2×1を、要らない2×1で割った数と同じである。
5×4×3×2×1は5!、2×1は2!。
で、2はどこから出てきたか、というと、5から3を取った余りだから5-3。
よって、5から3個取り出して並べる順列は5!÷(5-3)!ということになる。
一般項にすると、n個からr個を取り出して並べると、n!÷(n - r)!となる。
組合せは、この部分順列の応用で、取り出したものの並び方を問わない、というものだ。
即ち、部分順列の順列（並び方）分を割ったもの、ということになる。
rの順列はr!だから、n個からr個取る組み合わせは、n!÷(n-r)!÷r!つまりn!÷(r!×(n-r)!)となる。
＜／順列・組合せ＞

しかし、実際に10から3だけ選ぶ組合せであれば、

(10×9×8)÷(3×2×1)
=(10×3×4) ［約分］
=120

となり、一般項より随分簡単になる。
一般項を『暗記』してる人は、一般項を『理解』してる人より、確率計算で多分かなり損をしている筈。
一般項の公式なんて、人間が直接使うものではなく、コンピュータに食わせるものである。

・連勝複式
競馬で１着と２着を、順番を問わずに当てること。ちなみに16頭（普通何頭なのか知らないが、ネットで見たらとりあえず16頭だったので）で１，２着をランダムに選ぶなら、(16×15)÷(2×1)で、あたる確率は1/120。競馬やる人って、凄い度胸だなぁ、と。

・MAHAMANを使って
MAHAMANは、ご存じコンピュータRPGの草分けの一つWizardryの魔法使いの呪文。効果が完全にランダムなHAMANの上位バージョンで、７つの奇跡の中からランダムに選ばれた三つのうち一つを選んで効果が与えられる。ただし代償として、経験レベルがダウンする。ちなみに組み合わせは、(7×6×5)÷(3×2×1)で、35通りだが、望む奇跡が含まれている可能性については、階乗計算ではなく、単に3/7[=(1/7)+(6/7 * 1/6)+(6/7 * 5/6 * 1/5)]。

・RPGでモンスターのトレジャー確率とか
結局、繰りかえしが多いほど確率計算は有効なので、コンピュータRPGのように、同じ作業を何度もやるものでは、戦略的に確率を考えることが可能になる。裏返せば、それ以外に確率がそれほど有効な事例はあまり無かったりする。

・階乗計算をする方法は……
今号の特集「階乗計算博物館」参照のこと。

・大なり
日本語としてはいかがかと思うが A > B は『A大なりB』と読む。『AがBより大なり』という意味らしいが、相変わらず『足す』と同じで日本文法ではない。

・n = n - 1
手続き型プログラミング初心者の第一関門。コレを理解するには、評価のタイミングを理解しなければならない。逆に言えば、評価や代入を理解する良い機会。

・プログラミング言語によっては
推論型プログラミング言語の Prolog や、関数型プログラミング言語の Haskell では、そう扱われる。逆にそのような言語で、n = n - 1 のような矛盾した定義をすると、無限ループに陥るかエラーになる。

・手続き型(プログラミング)言語 procedural(programming)language
本来は、手続き(procedure)を抽象化できる機能を持った言語のことだが、単に処理を記述するタイプの言語を指して使われることが多く、オブジェクト指向プログラミング言語や、関数型プログラミング言語、そして推論型プログラミング言語に「それぞれ」対比され、「旧タイプ」というイメージで語られる（もっとも、手続きを抽象化できない言語は今となってはかえって珍しいので、どちらの定義でもさして変わらない）。関数型の元祖は Lisp、推論型の元祖は(おそらく)Prolog、そしてオブジェクト指向型の元祖は(Simula か)Smalltak だとすれば、1970 年代には全て出揃っていることになるが、現在に至ってもこの『手続き型』がプログラミング言語の王道であることに揺るぎは無い。しかし、コンピュータ能力の急速な発達により、スペックに余裕が出てきたため、様々なパラダイムの言語が実用的に使用でき、選択幅は間違いなく増えたと思われる。
ちなみに Python は手続き型がベースで、関数型とオブジェクト指向型を組み入れたマルチパラダイム言語だが、手続き型以外は全てオプションと公式にアナウンスされている（でも、Pascal のように完全に構造化されているわけでもなかったりする）

・「エス！ マム」
"Yes! ma'am."と言いたいらしい。ma'am は madam の中音消失。「はい、奥様」くらいの感じ。ちなみにイェスをエスと書くのは、筆者の趣味。

・整数の入力して下さい：1000
以下は本当に 1000!を Python で計算したものを掲載してある。

・ループを二重にすると……
昔は二重ループは文字検索のための必須アルゴリズムだったが、文字列検索機能を持つ LL ではほぼ無用。辞書のキーとアイテムを自動生成する時に使うぐらいか？

・（二重ループを抜けるために）例外処理を利用する
例外処理は、当然エラーの制御に使われるのだが、べつにエラーの制御だけを担当しなければならないわけではない。今回のように二重ループを抜ける手段として例外処理を使うのは、Python の通常のテクニックである（ただし、raise を使うこと。似ているからといって assert を使うコードがネットに上がっていたが、それは大間違いだ）。これは、GOTO よりもかなり面倒な書き方になるが、反面、制御構造を明確に出来るという利点がある。

・GOTO 文
シンプルなのだが、構造を考えないで使うとスパゲッティを招く。Python では最初から存在しない（以前は機能しない予約語としてこっそり残していたが、最近予約語を抹消された）。制御を任意のラベルにジャンプさせる場合は、以下のようなループを代替として使うなどの方法しかない。

```
while 1:
  if test1:
    ...
    continue
  elif test2:
    ...
    continue
  elif test3:
    ...
    continue
  ...
  else:
    break
```

上記の test で、カウンタなどの条件を調べれば、GOTO のように働かないこともない（強制であれば、例外処理の外にループを書く方法も無いわけでもないが……）

・なぜ例外処理を使う

詳細な理由は不明。おそらく、スライスなどを借用した ICON のエラーによる制御（イテレータ処理はまさにコレだ）を参考しにたと考えられるが、言及があった文章を、残念ながら見たことはない。

・オリジナルの例外

Excepiton クラスを継承した新しいクラスを定義する。

・〈for〉ループ

Python の for ループは便利だが、コレクションの知識が前提となる。

【Python あれこれ】２．ブロックとクラス定義

本文にも少し出てきましたが、Python の制御ブロックは、通常のプログラミング言語で言うところのブロックとは少し異なります。

たとえば for ループや while ループで、カウンタ専用の局所変数を作ろうとしてもできません。

内包表記を使えば別ですが、内包表記は式のみという条件があるので、やりたいことによっては、かなりトリッキーな見づらいコードになる可能性があります。

Python で名前空間を区切ろうと思ったら、モジュールに分ける、関数やクラス定義（定義ブロック）を使う、あるいは exec 関数を使う、などの方法に限定されます。

このうち、クラス定義はコードの中で名前空間を区切る一番手軽な方法ですが、class というキーワードがやや場違いかな（というか、実際本来の使用法ではないので当然ですが）ということと、名前空間用の名前（要するにクラス名）が余分に必要になることが気になります。

関数定義は一番きちんとしていますが、一度しか実行しないコードを関数にまとめて後で呼び出すのは、少し仰々しい気もしますし、クラスと同様、やはり関数名が残ります。

exec 関数による実行は、結果を一切残さないで実行できる点はいいのですが、文字列を一々つくって実行する煩わしさはダントツです。

モジュールは別ファイルに書く方法で、これも後腐れの少ない方法ですが、モジュールに記述されたコードに現在の実行環境からパラメータを渡すには、結局関数などを呼び出すしかない、という矛盾がありますし、第一、再利用しないならこれほど巨大な無駄に感じるものはないでしょう。

結局、邪魔にならない程度に変数を設定して、そのまま実行するか、あるいは若干の気持ち悪さを

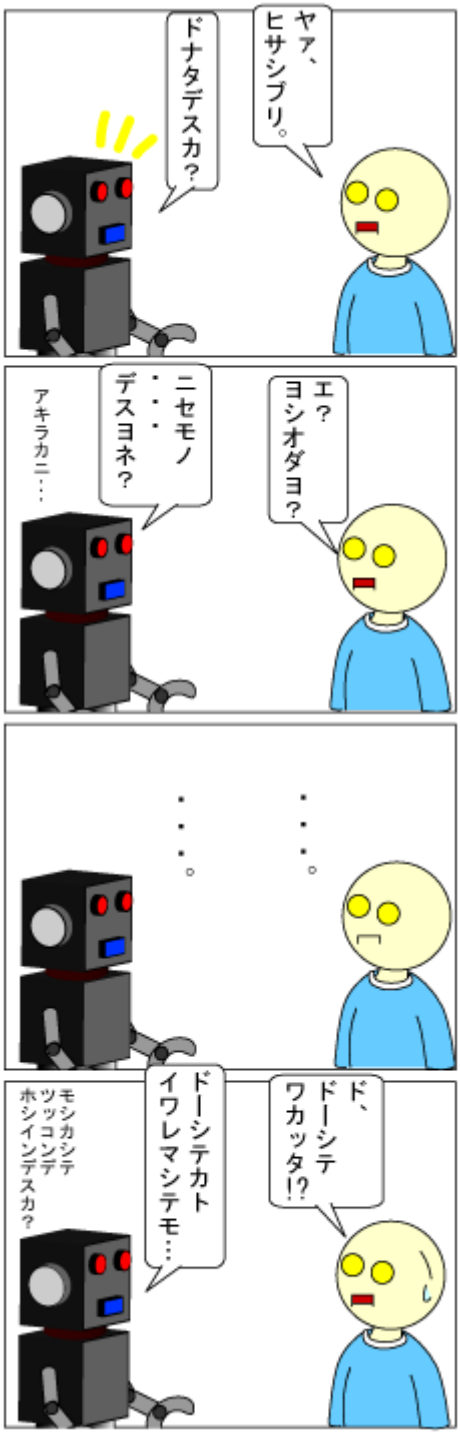
感じながらもクラスで一時的なブロックを作るのが、現実的な解決法になります。
なお、Pythonは3.0になる過程で、クラスの中の関数との関係が改良され、クラス名で中の関数を呼び出す際には、ただの関数（要するにstatic method）として呼び出せるようになりました（以前は『非束縛メソッド』という、使いにくいお約束があったのですが）
よって、クラスを関数のコンテナとして利用するのが楽になったわけですが、クラス内で関数を定義して、メソッドとしてではなく内部で使用する場合、スコープに注意してください。
クラス定義内変数（要するにクラス変数）は、クラス内で定義された関数の内側では『見えません』。
クラス内定義関数が見える『外側』とは、クラス内部をすっとばして『クラスの外』なのです（蛇足ながら言っておくと、クラス定義は関数定義ではないので、nonlocal宣言を行っても無駄です）
どうしてもクラス変数を参照したければ、引数として明示的に引き渡すか、クラスメソッドとして定義して、第一引数にクラスポインタとなる引数を追加することです。
厄介なルールですが、インスタンスを生成する構文と同居しているわけですので、仕方ないとあきらめてください。
しかしそれを差し引いても、独立名前空間でプログラミングし放題ですし、後からその結果もクラス属性として参照できます。
※ここまで書いておいてなんですが、C++やJavaを主に使う方には理解しづらいでしょうが、PythonではSmalltalkなどと同様にクラスはインスタンスを作らずともオブジェクトとして利用できます。
クラス定義ブロックによるローカルプログラミングは簡単でお勧めですので、一度試してみてください。

（機械伯爵）

よしおさんとロボ太

海鳥作

「ヨシオサンー登場」

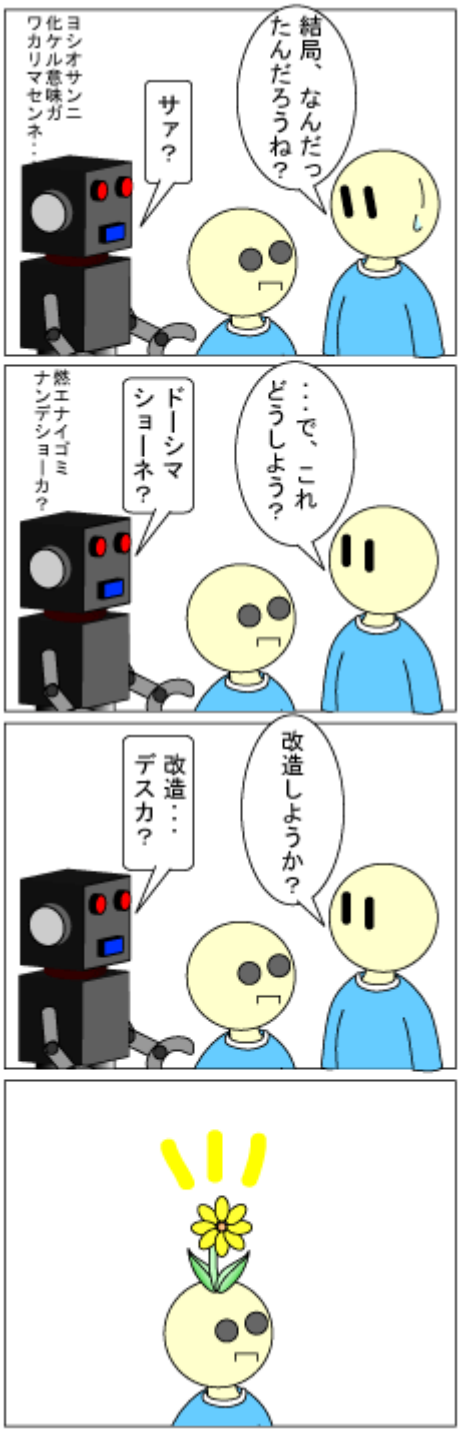


カタカナ会話。

「ヨシオサンー襲撃」

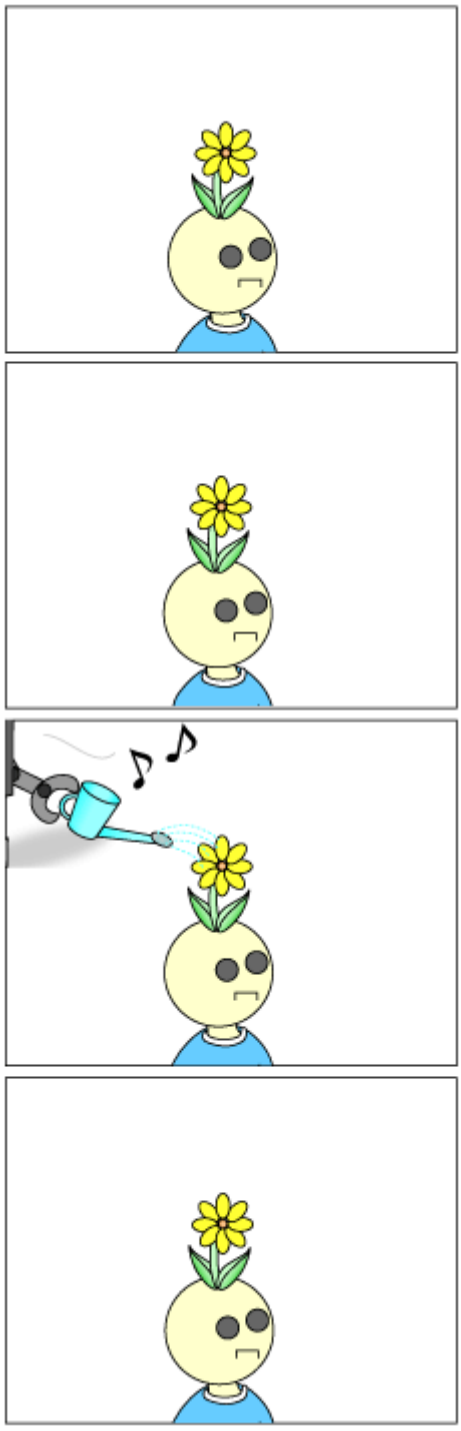
ドリル。

「ヨシオサンー転機」



エコロボジー。

「ヨシオサンー余生」



ガーデニングロボ。

ポストモダン・フレームワーク

Ruby on Rails、 Catalyst、そして、CakePHP

jscripter

1. はじめに

2005年のRuby on Railsの登場以来、Model-View-Controller(MVC)フレームワークの名が、我々市井のスクリプタにもよく聞こえてくるようになった。もちろん、MVCフレームワークは、Appleは早くも1996年にWebObjectsで実現していたし、Javaのプログラマにとってはは Struts(2001年)で御馴染みのものであった。Perlのスクリプタも Perlの MVCフレームワーク Maypole(2004年)を CPANや Simon Cozens氏の perl.comの記事で知っていたかもしれない。さらに言えば、本来、MVCフレームワークは知識表現のために構想されたものであった(Trygve Reenskaug、1979年)。MVCフレームワークは、PCの発展とともにGUIの基礎となったあと、インターネット時代にWebアプリケーションのフレームワークとして登場することは、HTMLの表現力の増大に伴う必然と言えるかもしれない。

著者は、1993年ぐらいには JPDBというテキストデータベースの Perlスクリプトを書いていた。コマンドラインでエスケープシーケンスを使って動作する原始的なものだった。1995年に Windows95が登場して以来、ワークステーションで開発されたGUIの世界が庶民の使うPCに浸透した。インターネットもほぼ同時に実用に近いものとなった。時は廻り、21世紀に入ってWebの発展は目覚ましいものとなり、あらゆる情報がWeb上に載るようになったのである。Web上の情報はGoogleなどの検索エンジンによってデータベース化され、誰もが容易に利用できるようになった。さらに2003年頃からRSSの配信が顕著になり、爆発的に広まった。Webは Semantic Webに変貌したのである。時を同じくして、著者はWebとデスクトップの情報を融合する方法についてCGIプログラミングによってささやかな提案をしてきた。Web上における知識表現についてさらに思索を深めることは、WebやPCなどのコンピューティング・デバイスの価値や意味を高めることにつながるだろうと思っている。

本稿は、「オブジェクト指向を廻る日記」(TSNETスクリプト通信1.1、2008年5月号)の続編のような位置づけで、知識表現の方法としてMVCフレームワークを調べることを目的とする。前稿はオブジェクト指向プログラミングと知識表現が結びついていることを明らかにした。本稿ではまずはMVCフレームワークに触れて、どのようなものかを知る機会としたい。

取り上げるものは、MVCフレームワークの名を広めたRuby on Rails、Perlの代表的なフレームワークとしてCatalyst、そして、最近PHPの世界で話題のCakePHPである。各システムをそれぞれインストールして、通常のプログラミングを実行してみる。何が違うのか。本稿では、ほとんどの一般ユーザーが使っているWindows環境へのインストールを試みる。

2. Ruby on Rails

Ruby on Railsの作者である David Heinemeier Hanssonが著者の一人である「Railsによるアジャイル Webアプリケーション開発」という Railsのガイドブックは現在、第3版(Agile Web Development with Rails Third Edition、以下AWDwR)が出ている。まだ英語版で、手元にあるのは、2009-4-7のバージョンである。先日届いたばかりだ。第1版が出たのは2005年だから、2年に一度のペースで更新されている。

Windows へのインストールには InstantRails-2.0(2007-12-28 リリースが最新)が例示されているのだが、Ruby 自体は 1.8.6 で問題ないだろうが、Rails のバージョンも 2.0.2 だし、少し古過ぎるのではないだろうか。RubyGems も 1.0.1 なので、Ruby ライブラリをアップグレードしていくためには古過ぎる。最新版にしては Windows に関しては手抜きの感じだ。InstantRails のサイトに 2009-03-06 に Windows で 1000 倍速い 1.9.1 対応の InstantRails を出すようにとの請願があったから今後は期待できるかもしれない。1.9.1 に移行するときはチェックしよう。

AWDwR では基本的には、Ruby は 1.8.7 を、Rails は RubyGems でインストールした 2.2.2 を推奨している。リレーショナルデータベース(RDBMS)としては SQLite3 を使う。

本稿では、次のような手順でインストールを行った。4 月下旬のインストールなので少し古くなっているが、現在の状況では最新版のインストールで問題は起きないと思われる。

2.1 ActiveScriptRuby のインストール

まず、Ruby をインストールしよう。Windows 環境で使うなら、ActiveScriptRuby を選択しよう。

COM Meets Ruby
http://arton.hp.infoseek.co.jp/indexj.html

上記のページから ActiveRuby.msi をダウンロードして、丁寧に説明されているインストール方法に従って、ActiveScriptRuby 1.8.7 をインストールする。msi という拡張子の Microsoft Installer を使うので簡単だ。本稿では、C:¥Ruby-1.8 にインストールした。

インストーラは PATH の設定は行わないので、[スタート]→[コントロールパネル]→[システム]→[詳細設定]→[環境変数]→[システム環境変数]→[変数]→[Path]→[編集]→[変数値の先頭に「C:¥Ruby-1.8¥bin;」を追加]→[OK]と設定しておく。

2.2 Rails のインストール

Rails は RubyGems でインストールする。RubyGems は Ruby のライブラリの標準的な管理システムになりつつある。

次のようにするのだが、これにはコマンドプロンプトの使い方がわかっている必要がある。後の作業にも必要だからおぼえておこう。[スタート]→[すべてのプログラム]→[アクセサリ]→[コマンドプロンプト]でコマンドプロンプトが起動する。本稿の場合は、C:¥Scripts¥Ruby のディレクトリでインストール作業をしている。コマンドプロンプトで、cd C:¥Scripts¥Ruby[Enter]とすれば、ディレクトリを移動できます。

下記の gem で始まる青文字の文字部分が、RubyGems による Rails のインストールの指示である。2.1.1 の Ruby のインストールで PATH の設定ができていないと動作しないので注意。青文字部分をキーボードから入力し、最後に[Enter]ボタンを押すとインストールがはじまる。

```
C:¥Scripts¥Ruby>gem install rails --include-dependencies
INFO:  `gem install -y` is now default and will be removed
```

```
INFO:  use --ignore-dependencies to install only the gems you list
Successfully installed activerecord-2.3.2
Successfully installed actionpack-2.3.2
Successfully installed actionmailer-2.3.2
Successfully installed activeresource-2.3.2
Successfully installed rails-2.3.2
5 gems installed
Installing ri documentation for activerecord-2.3.2...
Installing ri documentation for actionpack-2.3.2...
Installing ri documentation for actionmailer-2.3.2...
Installing ri documentation for activeresource-2.3.2...
Installing RDoc documentation for activerecord-2.3.2...
Installing RDoc documentation for actionpack-2.3.2...
Installing RDoc documentation for actionmailer-2.3.2...
Installing RDoc documentation for activeresource-2.3.2...
```

図: Rails のインストール結果(C:¥Ruby-1.8¥bin)

さて、RubyのPATHの中味を見てみよう。図1のように、C:¥Ruby-1.8¥binディレクトリにrails.batというRailsを起動するバッチがインストールされている。

2.3 Railsの起動とRubyGemsのインストール

次にデモを動かしてみよう。demoというアプリケーションを作る。次のようにworkディレクトリを作成して、workディレクトリに移動する。そして、rails demoとしてdemoアプリケーションを作成する。

```
C:¥Scripts¥Ruby>md work

C:¥Scripts¥Ruby>cd work


C:¥Scripts¥Ruby¥work>rails demo
      create
      create  app/controllers
      create  app/helpers
      create  app/models
      create  app/views/layouts
      create  config/environments
      create  config/initializers
      create  config/locales
      create  db
      create  doc
      create  lib
      create  lib/tasks
      create  log
      .....
      create  log/test.log
```

demoというディレクトリが作成されるので、demoディレクトリに移動する。

```
C:¥Scripts¥Ruby¥work>cd demo

C:¥Scripts¥Ruby¥work¥demo>dir
.....
2009/04/13  21:46    <DIR>          app
2009/04/13  21:46    <DIR>          config
2009/04/13  21:46    <DIR>          db
2009/04/13  21:46    <DIR>          doc
2009/04/13  21:46    <DIR>          lib
2009/04/13  21:46    <DIR>          log
2009/04/13  21:46    <DIR>          public
2009/04/13  21:46               307 Rakefile
2009/04/13  21:46            10,011 README
2009/04/13  21:46    <DIR>          script
2009/04/13  21:46    <DIR>          test
2009/04/13  21:46    <DIR>          tmp
2009/04/13  21:46    <DIR>          vendor
.....
```

Railsを動作させる。

```
C:¥Scripts¥Ruby¥work¥demo>ruby script/server
Rails requires RubyGems >= 1.3.1 (you have 1.2.0). Please `gem update --system`
and try again.
```

```
C:¥Scripts¥Ruby¥work¥demo>gem update --system
Updating RubyGems
Nothing to update
```

Rails 2.3.2(執筆時点のバージョン)はRubyGemsの1.3.1以上を要求する。したがって、新しいRailsを試すためには、InstantRails-2.0のままでは無理ということになる。ActiveScriptRuby 1.8.7もRubyGemsの1.2.0をインストールするので、エラーになる。ここでは、updateが効かないので、RubyGems 1.3.1をダウンロードし、解凍したディレクトリで、ruby setup.rbとしてインストールする。下記のサイトからダウンロードしよう。執筆時点では、バージョンは1.3.3となっている。

RubyForge: RubyGems: Project Info
http://rubyforge.org/projects/rubygems/

さて、再度トライ。

```
C:¥Scripts¥Ruby¥work¥demo>ruby script/server
=> Booting WEBrick
=> Rails 2.3.2 application starting on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2009-04-13 22:06:22] INFO  WEBrick 1.3.1
[2009-04-13 22:06:22] INFO  ruby 1.8.7 (2008-08-11) [i386-mswin32]
[2009-04-13 22:06:44] INFO  WEBrick::HTTPServer#start: pid=6612 port=3000
[2009-04-13 22:51:06] INFO  going to shutdown ... (Ctrl-Cでshutdown)
[2009-04-13 22:51:06] INFO  WEBrick::HTTPServer#start done.
Exiting
```

デフォルトでは、WEBrickというRubyに標準でインストールされているHTTPサーバーが動作する。終了するには、Ctrlキーを押さえながらCを押す。

終了する前に、Webブラウザで、http://localhost:3000/としてdemoを起動してみる。次のように表示されれば正常だ。

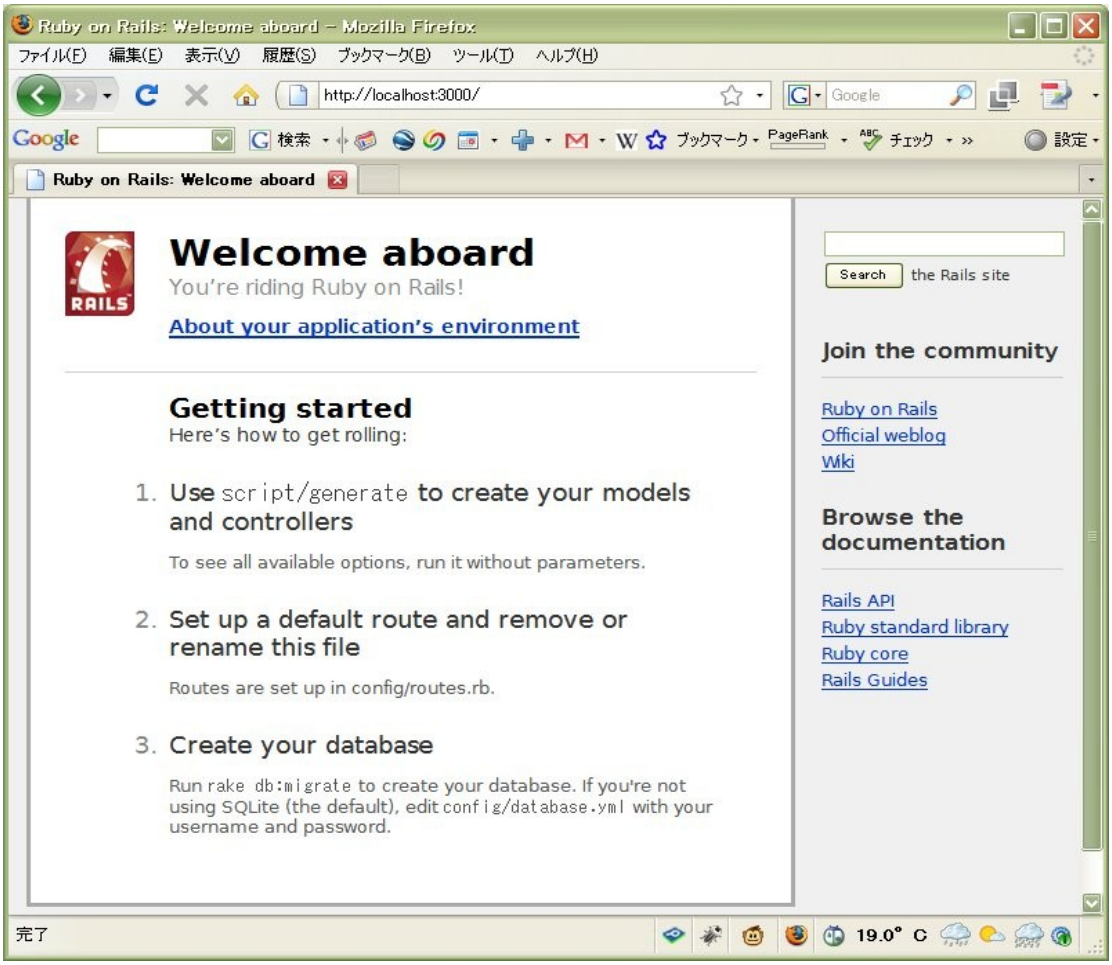


図: Railsの表示画面(http://localhost:3000/)

2.4 まだ足りないもの - SQLite3のインストール

まだ足りないもの^^;)がある。図2の画面で、About your application's environmentが表示されないのである。

SQLite3をインストールしよう。

SQLite Home Page
http://www.sqlite.org/
Precompiled Binaries For Windows
sqlitedll-3_6_13.zip (244.53 KiB)

解凍して、sqlite3.dllをパスの通ったディレクトリ、C:¥Ruby-1.8¥binなどにコピーする。執筆時点のバージョンは、sqlitedll-3_6_14_1.zip (244.93 KiB) である。

RubyForge: SQLite-Ruby: ファイルリスト
http://rubyforge.org/frs/?group_id=254

上記のサイトからsqlite3-ruby-1.2.3-mswin32.gemをダウンロードして次のようにインストールす

る。最新版にはmswin32版がないためにオンラインインストールは失敗してしまう。

```
C:¥Scripts¥Ruby>gem install sqlite3-ruby-1.2.3-mswin32.gem
Successfully installed sqlite3-ruby-1.2.3-x86-mswin32
1 gem installed
Installing ri documentation for sqlite3-ruby-1.2.3-x86-mswin32...
Installing RDoc documentation for sqlite3-ruby-1.2.3-x86-mswin32...
```

これでインストールは終了だ。ご苦労様。再度、サーバーを起動してアクセスしてみよう。

図: Railsの表示画面(About your application's environment)

インストールしたすべてのソフトウェアが表示される。

このようにインストールを一通り体験してみると、Instant-Railsの必要性がよくわかる。スクリプティング言語あるいはコマンドプロンプト初心者には難しいかもしれない。しかし、この関門をクリアできないようでは使えるレベルには到達できないかもしれない。あらかじめ、RubyGemsとSQLite3のインストールは済ませておいて、Railsをインストールするほうが効率がよいだろう。

次にコマンドプロンプトでサーバーが動作する様子を示しておこう。

図: Railsのコマンドプロンプト起動画面

2.5 Railsでプログラミング

本稿ではHello worldを動かすかわりに、簡単なスクレーピング・スクリプティングを試してみよう。それを通じて、MVCフレームワークとは何なのかを体験する。Modelはまだ使わないので、ControllerとViewについて見る。

まず、demoアプリケーションを作ったように、workディレクトリで、rails rssとして、rssアプリケーションを作成する。¥work¥rss¥app¥controllerディレクトリに次のrss_controller.rbを置く。これがControllerである。アプリケーションを作る過程で、同じディレクトリにあらかじめapplication_controller.rbが作られているはずだ。

[¥work¥rss¥app¥controller¥rss_controller.rb]

```
require 'rubygems'
require 'nokogiri'
require 'open-uri'

class RssController < ApplicationController
	def title
		t = []
		@baseurl = 'http://homepage1.nifty.com/kazuf/'
		@xml = 'renewal.xml'
		d = Nokogiri::HTML(open(@baseurl + @xml))
```

```
            d.xpath( '//item/title' ).each do |e|
                  t.push(e.inner_text)
            end
            @titles = t
      end
end
```

Viewは、¥work¥rss¥app¥viewディレクトリにrssというディレクトリを作って、次のようなerbテンプレートを置くことに対応する。ERbは、Rubyのコードが埋め込まれたhtmlを解釈することができるのだが、.html.erbという拡張子はRailsにERbを使うように知らせる仕組みになっている。.rhtmlの拡張子でも同様に動作するようだ。Rails本の第1版では.rhtmlが使われていた。

[¥work¥rss¥app¥view¥rss¥title.html.erb]

```
<html>
      <head>
            <title>RSS タイトルリスト</title>
      </head>
      <body>
            <h1>RSS タイトルリスト</h1>
            <table cellpadding="5" cellspacing="0">
                  <tr><td><h3>RSS: <%= @baseurl + @xml %></h3></td></tr>
                  <tr><td><b>タイトル</b></td></tr>
            <% for title in @titles %>
                  <tr valign="top">
                        <td><%= title %></td>
                  </tr>
            <% end %>
            </table>
      </body>
</html>
```

¥work¥rssディレクトリで、ruby script/serverとしてRailsサーバーを起動する。上記のrssアプリケーションにアクセスしてみよう。http://localhost:3000/rss/titleとする。

図: Railsのrssアプリケーション表示(http://localhost:3000/rss/title)

著者の更新日記「日曜プログラマのひとりごと」のrssフィードのitemのtitleがリストアップされたはずである。上記のURLのパスの「/rss/title」は「/コントローラ名/メソッド名」に対応している。メソッドの出力はviewディレクトリに格納された¥rss¥title.html.erbテンプレートに従って表示される。「/rss/title」がViewの「/テンプレートのあるディレクトリ/.html.erbテンプレートファイルの名前」に対応していることにも注意しておこう。ControllerとViewは対応関係にある。Ruby on Railsの場合、コントローラのメソッドはアクションと呼ばれる。

メソッドを書くときに注意するのは、メソッドの外で参照する変数には、インスタンス変数を使うこと。インスタンス変数は「@」を頭に持つ変数である。

rss_controller.rbは、Nokogiriというスクレーピング(Webに置かれたHTML/XMLなどのデータから必要な部分を取り出す)用のライブラリで、RSSフィードからXpathを使ってitem主要素のtitle要素を取り出している。

この例ではMVCフレームワークとしてはModelを使っていないが、リストが表示されるのだからデータが構造を持っていることは明白で、そのモデルはRSS(RDF Site Summary、Rich Site Summary、あるいはReally Simple Syndication)ということになる。しかし、本稿では、Modelには立ち入らず、次稿で取り扱うことにしよう。

3. Catalyst

Ruby on RailsとPerlのMVCフレームワークであるCatalystを比較してみよう。同じMVCフレームワークでも何か違いがあるのだろうか。Catalystは、Simon Cozens氏のMaypoleから派生すると同時に、Ruby on Railsからもインスパイアされている。

3.1 ActivePerlのインストール

PerlはActiveState社のActivePerl 5.8.9を使う。下記のURLから、普通のWindowsユーザー(32ビットのXP/Vista)であれば、Windows (x86)用の Windows Installer (MSI) を入手しよう。

http://www.activestate.com/activeperl/downloads/

ActiveScriptRubyと同様にWindowsインストーラーなので、ダブルクリックなり、右クリック→[インストール]でOKである。本稿では、C:¥Perl5.8にインストールしている。「C:¥Perl5.8¥bin;」を環境変数Pathに追加しておくこと。

3.2 Catalystのインストール

ActivePerlは、Perl Package ManagerというGUIアプリケーションを持っているのでライブラリのインストールやバージョン管理に大変便利である。[スタート]→[すべてのプログラム]→[ActivePerl 5.8.9]→[Perl Package Manager](以下、PPM)で起動する。

PPMの[Edit]→[Preferences]→[Repositories]→[Suggested: Select from list ...]から、ライブラリのレポジトリを選択・追加しよう。ActivePerlの正規のレポジトリ以外に、ppm.tcool.orgとtheoryx5.uwinnipeg.caは必須である。レポジトリはPPM用ライブラリパッケージのデータベースである。

大変申し訳ないのだが、ライブラリとして何が必須かなんだが、適当に入れてしまったので、はっきりしたことはわからない。それでも動くから、主要なものと思われるものを入れていけばよい。Catalyst関連のライブラリは山ほどあるが、マニュアルを見るかぎりは、次の二つを入れておけば動くと思われる。

1. Catalyst-Runtime
2. Task-Catalyst

最初のものは、catalyst.batを含むCatalystフレームワークのライブラリである。後者は、Catalystが依存関係のある23のライブラリをインストールするためのライブラリである。(+)ボタン(Mark for Install)でマークして、緑色の矢印(Run marked actions)でインストールを開始する。

catalyst.batを簡単に起動できるように、「C:¥Perl5.8¥site¥bin;」を環境変数Pathに追加しておこう。そして、このパスにcatalyst.batがあることを確認しておこう。そうすればインストールは成功である。

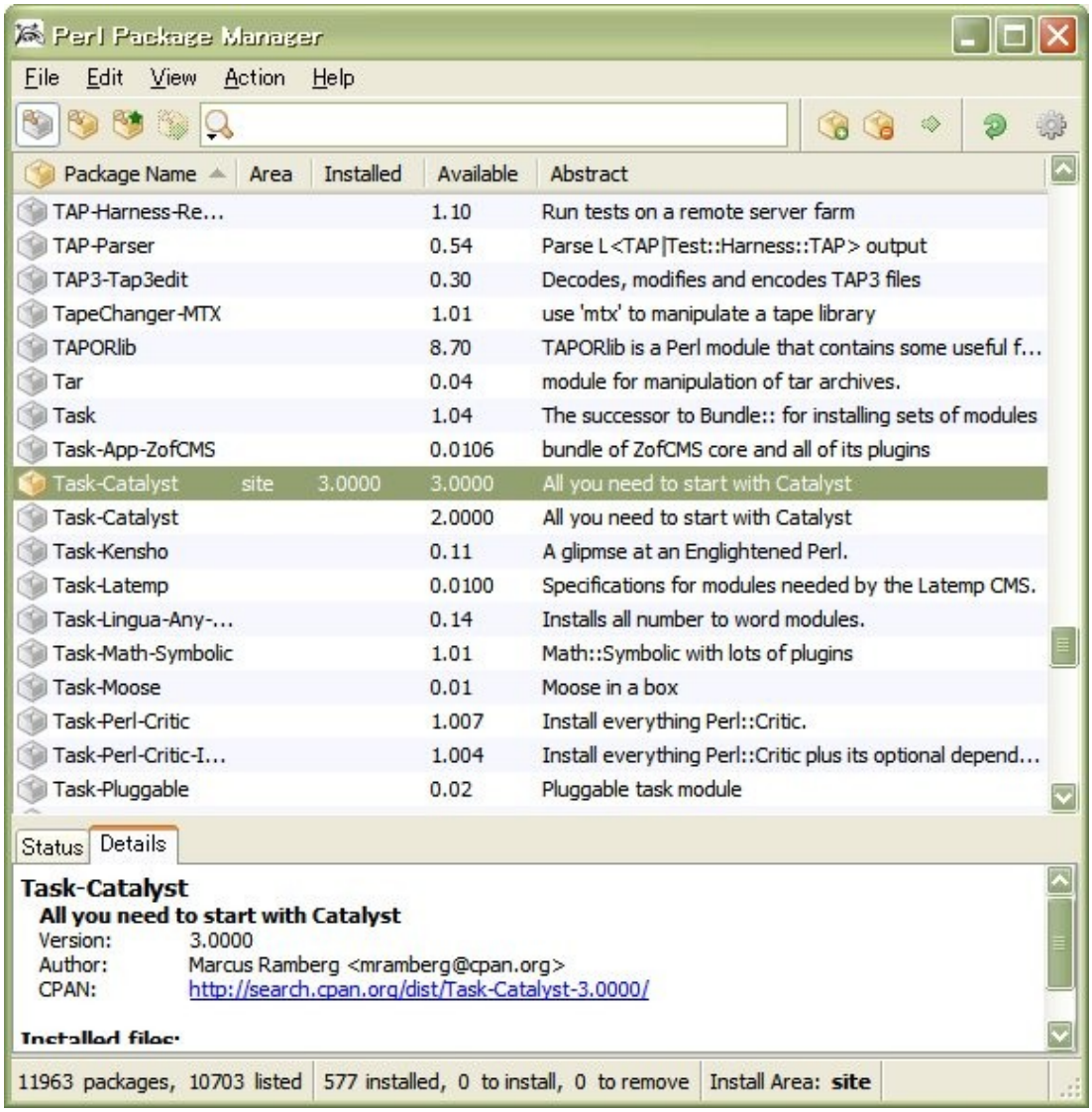


図: Perl Package Manager

3.3 Catalystアプリケーションの作成と起動

Railsで作ったのと同様のRSSフィードをスクレーピングするアプリケーションを作成する。

catalyst.batでRssアプリケーションを生成する。適当なディレクトリで、次のようにする。

```
C:¥Scripts¥Perl¥Catalyst>catalyst Rss
created "Rss"
created "Rss¥script"
created "Rss¥lib"
created "Rss¥root"
created "Rss¥root¥static"
created "Rss¥root¥static¥images"
```

```
created "Rss¥t"
created "Rss¥lib¥Rss"
created "Rss¥lib¥Rss¥Model"
created "Rss¥lib¥Rss¥View"
created "Rss¥lib¥Rss¥Controller"
created "Rss¥rss.conf"
created "Rss¥lib¥Rss.pm"
created "Rss¥lib¥Rss¥Controller¥Root.pm"
created "Rss/README"
created "Rss/Changes"
created "Rss¥t/01app.t"
created "Rss¥t/02pod.t"
created "Rss¥t/03podcoverage.t"
created "Rss¥root¥static¥images¥catalyst_logo.png"
created "Rss¥root¥static¥images¥btn_120x50_built.png"
created "Rss¥root¥static¥images¥btn_120x50_built_shadow.png"
created "Rss¥root¥static¥images¥btn_120x50_powered.png"
created "Rss¥root¥static¥images¥btn_120x50_powered_shadow.png"
created "Rss¥root¥static¥images¥btn_88x31_built.png"
created "Rss¥root¥static¥images¥btn_88x31_built_shadow.png"
created "Rss¥root¥static¥images¥btn_88x31_powered.png"
created "Rss¥root¥static¥images¥btn_88x31_powered_shadow.png"
created "Rss¥root¥favicon.ico"
created "Rss/Makefile.PL"
created "Rss¥script/rss_cgi.pl"
created "Rss¥script/rss_fastcgi.pl"
created "Rss¥script/rss_server.pl"
created "Rss¥script/rss_test.pl"
created "Rss¥script/rss_create.pl"
Change to application directory and Run "perl Makefile.PL" to make sure your ins
tall is complete
```

指示に従って、アプリケーション・ディレクトリに移動して、インストールを完了させる。

```
C:¥Scripts¥Perl¥Catalyst>cd Rss

C:¥Scripts¥Perl¥Catalyst¥Rss>perl Makefile.PL
include C:/Scripts/Perl/Catalyst/Rss/inc/Module/Install.pm
include inc/Module/Install/Metadata.pm
include inc/Module/Install/Base.pm
Cannot determine perl version info from lib¥Rss.pm
include inc/Module/Install/Catalyst.pm
*** Module::Install::Catalyst
include inc/Module/Install/Makefile.pm
Please run "make catalyst_par" to create the PAR package!
*** Module::Install::Catalyst finished.
```

```
include inc/Module/Install/AutoInstall.pm
include inc/Module/Install/Include.pm
include inc/Module/AutoInstall.pm
*** Module::AutoInstall version 1.03
*** Checking for Perl dependencies...
[Core Features]
- Catalyst::Runtime                ...loaded. (5.80003 >= 5.80003)
- Catalyst::Plugin::ConfigLoader   ...loaded. (0.20)
- Catalyst::Plugin::Static::Simple ...loaded. (0.20)
- Catalyst::Action::RenderView     ...loaded. (0.08)
- parent                           ...loaded. (0.221)
- Config::General                  ...loaded. (2.42)
*** Module::AutoInstall configuration finished.
include inc/Module/Install/WriteAll.pm
Writing META.yml
include inc/Module/Install/Win32.pm
include inc/Module/Install/Can.pm
include inc/Module/Install/Fetch.pm
Writing Makefile for Rss
```

ここで、アプリケーション・サーバーは起動できるのだが、先に進もう。

rss_create.pl というヘルパー・スクリプトを使って、controller(コントローラ)スクリプト Item.pm を作る。

```
C:¥Scripts¥Perl¥Catalyst¥Rss>perl script¥rss_create.pl controller Item
 exists "C:¥Scripts¥Perl¥Catalyst¥Rss¥lib¥Rss¥Controller"
 exists "C:¥Scripts¥Perl¥Catalyst¥Rss¥t"
created "C:¥Scripts¥Perl¥Catalyst¥Rss¥lib¥Rss¥Controller¥Item.pm"
created "C:¥Scripts¥Perl¥Catalyst¥Rss¥t¥controller_Item.t"
```

次に、view(ビュー)スクリプト TT.pm を作る。最初の TT がスクリプト名の指定、後者の TT が Template Toolkit の指定である。

```
C:¥Scripts¥Perl¥Catalyst¥Rss>perl script¥rss_create.pl view TT TT
 exists "C:¥Scripts¥Perl¥Catalyst¥Rss¥lib¥Rss¥View"
 exists "C:¥Scripts¥Perl¥Catalyst¥Rss¥t"
created "C:¥Scripts¥Perl¥Catalyst¥Rss¥lib¥Rss¥View¥TT.pm"
created "C:¥Scripts¥Perl¥Catalyst¥Rss¥t¥view_TT.t"
```

Item.pmにメソッド(アクション)を書き込もう。RSSフィードのitem主要素のtitle要素のみを出力するメソッドtitleと、title要素にlink要素のURLを使ってハイパーリンクを張り、description要素もpタグで括って出力するallというメソッドの二つを作成した。

[¥Rss¥lib¥Rss¥Controller¥Item.pm]

```
package Rss::Controller::Item;

use strict;
use warnings;
use parent 'Catalyst::Controller';
use Hasami::Rss;

=head1 NAME

Rss::Controller::Item - Catalyst Controller

=head1 DESCRIPTION

Catalyst Controller.

=head1 METHODS

=cut

sub title : Local {
    my ( $self, $c ) = @_;

       my $hasami = Hasami::Rss->new(
                           url => 'http://homepage1.nifty.com/kazuf/renewal.xml',
                     );
       $hasami->getxml;
       my @titles = $hasami->getvalues(0);# xpath = '//item/title'
       $c->stash->{titlelist} = ¥@titles;# 配列のリファレンスを渡す
       $c->stash->{template} = 'item/title.tt';# テンプレートの指定
}

sub all : Local {
    my ( $self, $c ) = @_;

       my $hasami = Hasami::Rss->new(
                           url => 'http://homepage1.nifty.com/kazuf/renewal.xml',
                     );
       $hasami->getxml;
       my @titles = $hasami->getvalues(0);# '//item/title'
       my @urls = $hasami->getvalues(1);# '//item/url'
       my @descriptions = $hasami->getvalues(2);# '//item/description'
       my @items = ();
       for(my $i=0;$i<=$#titles;$i++){
              push(@items,
               "<li><a href=¥"$urls[$i]¥">$titles[$i]</a><p>$descriptions[$i]</p>");
```

```
        }
        $c->stash->{itemlist} = ¥@items;# 配列のリファレンスを渡す
        $c->stash->{template} = 'item/all.tt';# テンプレートの指定
}

=head2 index

=cut

sub index :Path :Args(0) {
    my ( $self, $c ) = @_;

    $c->response->body('Matched Rss::Controller::Item in Item.');
}


=head1 AUTHOR

A clever guy

=head1 LICENSE

This library is free software. You can redistribute it and/or modify
it under the same terms as Perl itself.

=cut

1;
```

著者の項の、A clever guyは笑わせるが、ドキュメント化も想定したモジュールの作り方の見本の枠組みも提供される。

メソッド(アクション)は特別な属性を持つサブルーチンである。Localという属性を持つ場合、例えば、package名が Rss::Controller::Itemで、sub title : Local {...}のアクションの場合、

http://localhost:3000/item/title

にマッチする。各種の属性については、Catalyst::Manual::IntroのAction typesの項を参照しよう。

Catalystの$c->stashメソッドは、データを蓄積し、それをコンポーネント間で手渡すことができる。上記の例では、templateキーでテンプレートのパスを指定し、テンプレートで使う変数をキーにして、変数のリファレンスを渡すことができる。他のメソッドについては、Catalystのマニュアルを参照しよう。

具体的には、次のような二つのテンプレートをRssプリケーションのrootディレクトリに配置する。

[¥Rss¥root¥item¥title.tt]

```
<ul>
[% FOREACH title IN titlelist %]
<li>[% title %]
[% END %]
</ul>
```

[¥Rss¥root¥item¥all.tt]

```
<ul>
[% FOREACH item IN itemlist %]
[% item %]
[% END %]
</ul>
```

さて、Rssアプリケーションを起動してみよう。

```
C:¥Scripts¥Perl¥Catalyst¥Rss>perl script¥rss_server.pl
[debug] Debug messages enabled
[debug] Statistics enabled
[debug] Loaded plugins:
.----------------------------------------------------------------------------.
| Catalyst::Plugin::ConfigLoader  0.20                                       |
| Catalyst::Plugin::Static::Simple  0.20                                     |
'----------------------------------------------------------------------------'

[debug] Loaded dispatcher "Catalyst::Dispatcher"
[debug] Loaded engine "Catalyst::Engine::HTTP"
[debug] Found home "C:¥Scripts¥Perl¥Catalyst¥Rss"
[debug] Loaded Config "C:¥Scripts¥Perl¥Catalyst¥Rss¥rss.conf"
[debug] Loaded components:
.-----------------------------------------------------------------+----------.
| Class                                                           | Type     |
+-----------------------------------------------------------------+----------+
| Rss::Controller::Item                                           | instance |
| Rss::Controller::Library::Login                                 | instance |
| Rss::Controller::Root                                           | instance |
| Rss::View::TT                                                   | instance |
'-----------------------------------------------------------------+----------'

[debug] Loaded Private actions:
.----------------------+--------------------------------------+--------------.
| Private              | Class                                | Method       |
+----------------------+--------------------------------------+--------------+
| /default             | Rss::Controller::Root                | default      |
```

```
| /end                | Rss::Controller::Root               | end         |
| /index              | Rss::Controller::Root               | index       |
| /item/index         | Rss::Controller::Item               | index       |
| /item/title         | Rss::Controller::Item               | title       |
| /item/all           | Rss::Controller::Item               | all         |
| /library/login/index | Rss::Controller::Library::Login    | index       |
'---------------------+-------------------------------------+-------------'

[debug] Loaded Path actions:
.-----------------------------------+-------------------------------------.
| Path                              | Private                             |
+-----------------------------------+-------------------------------------+
| /                                 | /default                            |
| /                                 | /index                              |
| /item                             | /item/index                         |
| /item/all                         | /item/all                           |
| /item/title                       | /item/title                         |
| /library/login                    | /library/login/index                |
'-----------------------------------+-------------------------------------'

[info] Rss powered by Catalyst 5.80003
You can connect to your server at http://dell3:3000
[info] *** Request 1 (0.011/s) [756] [Fri May 29 15:57:47 2009] ***
[debug] "GET" request for "site/test" from "127.0.0.1"
[debug] Arguments are "site/test"
[info] Request took 0.023715s (42.167/s)
.-----------------------------------------------------------------+----------.
| Action                                                          | Time     |
+-----------------------------------------------------------------+----------+
| /default                                                        | 0.000536s |
| /end                                                            | 0.002432s |
'-----------------------------------------------------------------+----------'

[info] *** Request 2 (0.020/s) [756] [Fri May 29 15:57:54 2009] ***
[debug] "GET" request for "/" from "127.0.0.1"
[info] Request took 0.018201s (54.942/s)
.-----------------------------------------------------------------+----------.
| Action                                                          | Time     |
+-----------------------------------------------------------------+----------+
| /index                                                          | 0.001484s |
| /end                                                            | 0.001833s |
'-----------------------------------------------------------------+----------'

[info] *** Request 4 (0.035/s) [756] [Fri May 29 15:58:09 2009] ***
[debug] "GET" request for "item/title" from "127.0.0.1"
[debug] Path is "item/title"
[debug] Rendering template "item/title.tt"
[info] Request took 0.617309s (1.620/s)
.-----------------------------------------------------------------+----------.
```

```
| Action                                                          | Time      |
+-----------------------------------------------------------------+-----------+
| /item/title                                                     | 0.521261s |
| /end                                                            | 0.083054s |
|  -> Rss::View::TT->process                                      | 0.079788s |
'-----------------------------------------------------------------+-----------'

[info] *** Request 5 (0.041/s) [756] [Fri May 29 15:58:18 2009] ***
[debug] "GET" request for "item/all" from "127.0.0.1"
[debug] Path is "item/all"
[debug] Rendering template "item/all.tt"
[info] Request took 0.652954s (1.532/s)
.-----------------------------------------------------------------+-----------.
| Action                                                          | Time      |
+-----------------------------------------------------------------+-----------+
| /item/all                                                       | 0.625000s |
| /end                                                            | 0.009914s |
|  -> Rss::View::TT->process                                      | 0.007491s |
'-----------------------------------------------------------------+-----------'
```

それでは、Webブラウザからアクセスして表示させてみよう。

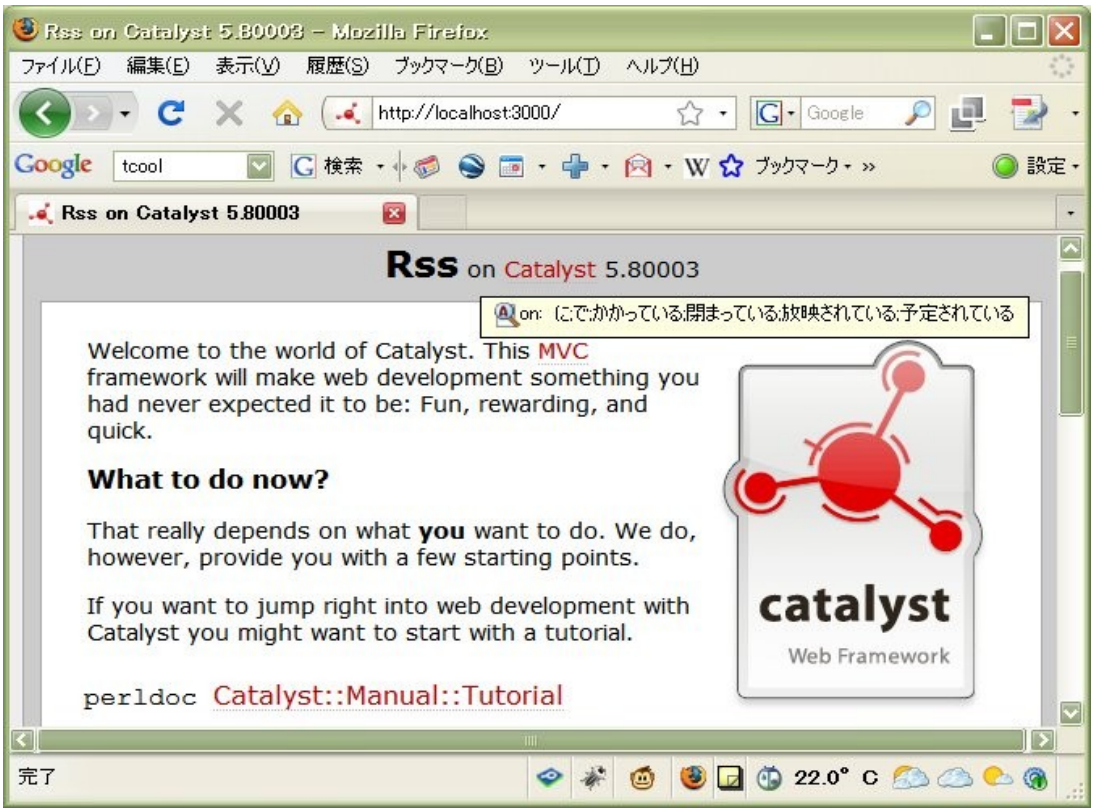


図: CatalystのRssアプリケーションの表示(http://localhost:3000/)

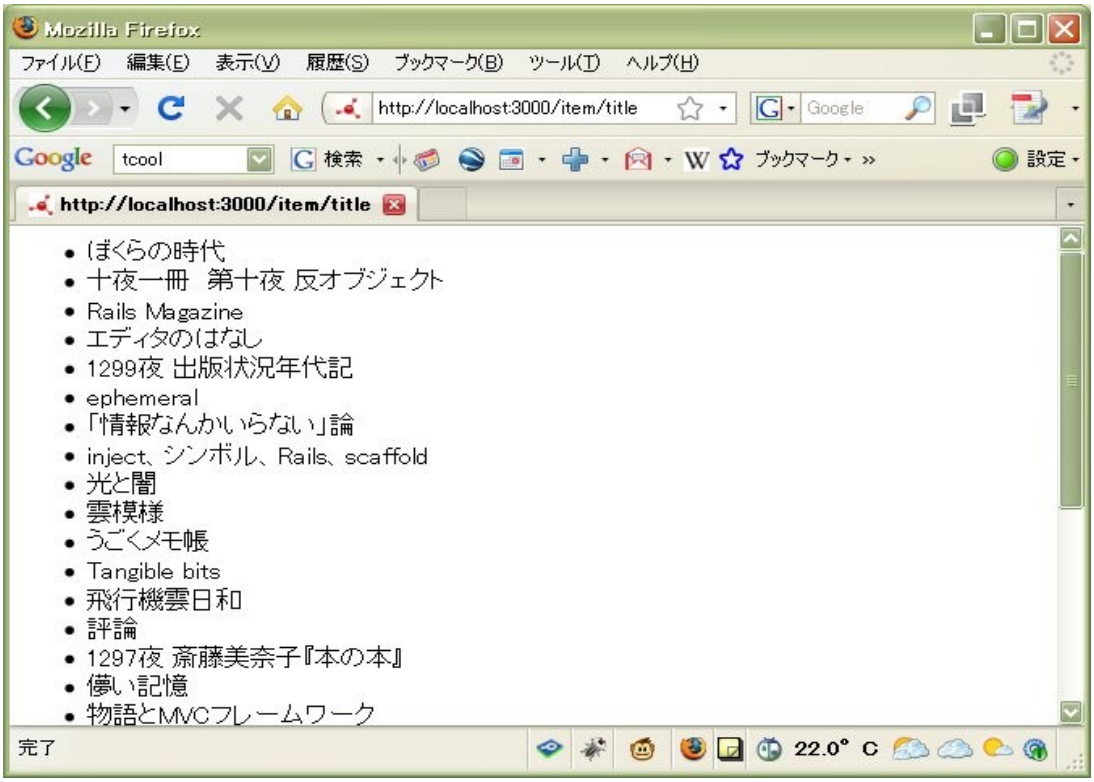


図: CatalystのRssアプリケーションの表示(http://localhost:3000/item/title)

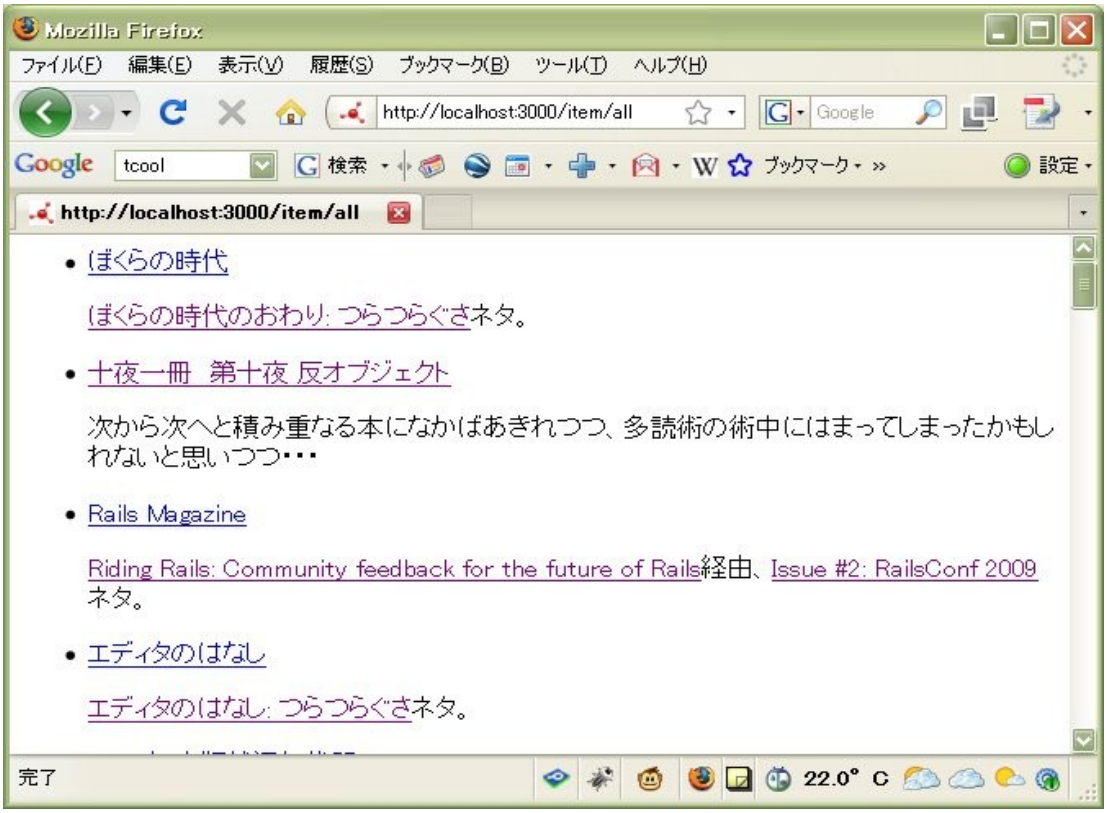


図: CatalystのRssアプリケーションの表示(http://localhost:3000/item/all)

3.4 CatalystとRailsの形式的な比較

Railsはコントローラのアクションの出力をERbでHTMLに埋め込むのに対して(ERbテンプレートのほかにBuilderテンプレートも使える)、CatalystもアクションをTemplate Toolkitなどのテンプレートに結びつける。アクションを起こすために、URLのパスに「コントローラ名/アクション名」を使うのも一緒である。しかし、それを実現している方法はデイレクトリの構成などから考えてもかなり違いがありそうだ。

図: Catalystアプリケーションがインストールされたディレクトリ

Catalystは、libデイレクトリにController、Model、Viewの三つのディレクトリがあり、ここでアプリケーションを書くことになる。Controllerディレクトリにあるitem.pmのファイル名がコントローラ名である。Viewディレクトリには、TT.pmがある。テンプレートは、rootディレクトリにパスに従って配置する。

図: Railsのrssアプリケーションのインストールデイレクトリ

Railsは、基本的には、appディレクトリにあるControllers、Models、Helpers、Viewsディレクトリにアプリケーションを格納する。コントローラ名は、Controllerディレクトリにあるファイル名の先頭に書いて区別する。テンプレートは、Viewsディレクトリにパスに従って配置する。

さて、MVCフレームワークとは関係ないが、RailsではNokogiriというスクレーピング用ライブラリを使った。同様のことをCatalystで実現するために、XML::XPathというPerlのライブラリを使って、Hasami::Rssというご都合主義的ライブラリを作成して使っている。Mooseというオブジェクト指向プログラミング用モジュールを使用している。参考までに載せておく。

[C:¥Perl5.8¥site¥lib¥Hasami¥Rss.pm]

```
package Hasami::Rss;
use Moose;
use LWP::Simple;
use XML::XPath;
use HTML::Entities;
use Encode;

has 'url' => (is => 'rw', isa => 'Str', default => '');# a url of XML-target
has 'tempfile' => (is => 'rw', isa => 'Str', default => 'hasami_temp.xml');# temporary file
has 'xpath' => (is => 'ro', isa => 'ArrayRef',
 default => sub { ['//item/title','//item/link','//item/description'] });# itemの副要素
has 'encoding' => (is => 'rw', isa => 'Str', default => 'utf-8');# utf-8 encoding

sub getxml {
```

```
	my $self = shift;
	my $rc = getstore($self->url, $self->tempfile);# an LWP::Simple method
	if($rc != 200){
		print $rc,": Not found or ...¥n";
		exit;
	}else{
		return 1;
	}
}

sub getvalues {
	my ($self, $elnum) = @_;
	my @values = ();
	my $value = "";
	my $xp = XML::XPath->new(filename => $self->tempfile);
	my $nodes = $xp->find($self->xpath->[$elnum]);
	foreach my $node ($nodes->get_nodelist) {
	($value = encode($self->encoding, $node->toString)) =~ s/<.+?>(.*)<¥/.+?>/$1/;
		push(@values, decode_entities($value));
	}
	return @values;
}

1;
```

---

4. CakePHP

ようやく、最後の作業に辿りついた。CakePHPである。PHP人気は衰えるどころか、CakePHPの登場によってさらに燃え盛るかという感じなんだろうなあ。

4.1 PHP/Apacheのインストール

CakePHPを使うためには、PHPとApacheが必要である。既にインストールしている人は問題ない。

PHPのインストールもMicrosoftインストーラなので簡単だ。URLから、Windows用のインストーラをダウンロードして、インストールしよう。Apacheのインストール先を指定できるので、先にApacheをインストールしておくこと。httpd.confをPHPを使えるように書き換えてくれる。

PHP: Downloads
http://www.php.net/downloads.php

Apacheは、apache_2.2.11-win32-x86-openssl-0.9.8i.msiなどを下記URLなどからダウンロードしてインストールしておこう。

Index of /mirror/apache/httpd/binaries/win32

http://www.meisei-u.ac.jp/mirror/apache/httpd/binaries/win32/

4.2 CakePHPのインストール

下記URLからダウンロードして、インストールしよう。Lhaca+デラックスなどで解凍して、中味をApacheのhtdocsディレクトリにcake¥sampleディレクトリ(本稿の設定)などを作成してコピーする。

CakePHP: 高速開発 php フレームワーク。 Home
http://cakephp.jp/

Apacheのhttpd.conf(インストールディレクトリのconfディレクトリにある)の「LoadModule rewrite_module modules/mod_rewrite.so」の行の先頭が「#」でコメントされていれば、削除して、rewrite_moduleを有効にする。さらに、.htaccessの設定を有効にするために、

```
<Directory "C:/anhttpd/htdocs/cake">
    AllowOverride All
</Directory>
```

のように、httpd.confに付け加える。掌田津耶乃著「CakePHPによるWebアプリケーション開発」(秀和システム、2009年)には「Allow from all」と書かれているが不要だし、そのままでは動作しない。この本では、Apache/PHPのインストールにXAMPP(Apache、MySQL、PHP、Perlなど)パッケージを使っている。

4.3 CakePHPのアプリケーション作成・起動

Pukiwikiのユーザー経験は長いのだが、PHPでアプリケーションは作ったことがない。前記のCake本の2-1.シンプルなWebアプリケーションをなぞり、そこにRSSフィードのスクレーピング・アクションを付け加えてみた。

[¥htdocs¥cake¥sample¥controllers¥hello_controller.php]]

```
<?php

class HelloController extends AppController {

	public $name = 'hello';
	public $uses = null;
	public $autoLayout;
	public $autoRender;

	function index(){
		$this->autoLayout = true;
		$this->autoRender = true;
	}

	function other(){
```

```
		$this->autoLayout = false;
		$this->autoRender = true;
		echo "これは、index以外の表示です。";
	}

	function rss(){
		$this->autoLayout = false;
		$this->autoRender = true;
		$url = "http://homepage1.nifty.com/kazuf/renewal.xml";
		$str = file_get_contents($url);
		$doc = new DOMDocument();
		@$doc->loadHTML($str);
		$xpath = new DOMXPath($doc);
		$arts = $xpath->query("//item/title");

		foreach ($arts as $art){
		echo $art->nodeValue."<br />";
		}
	}
}

?>
```

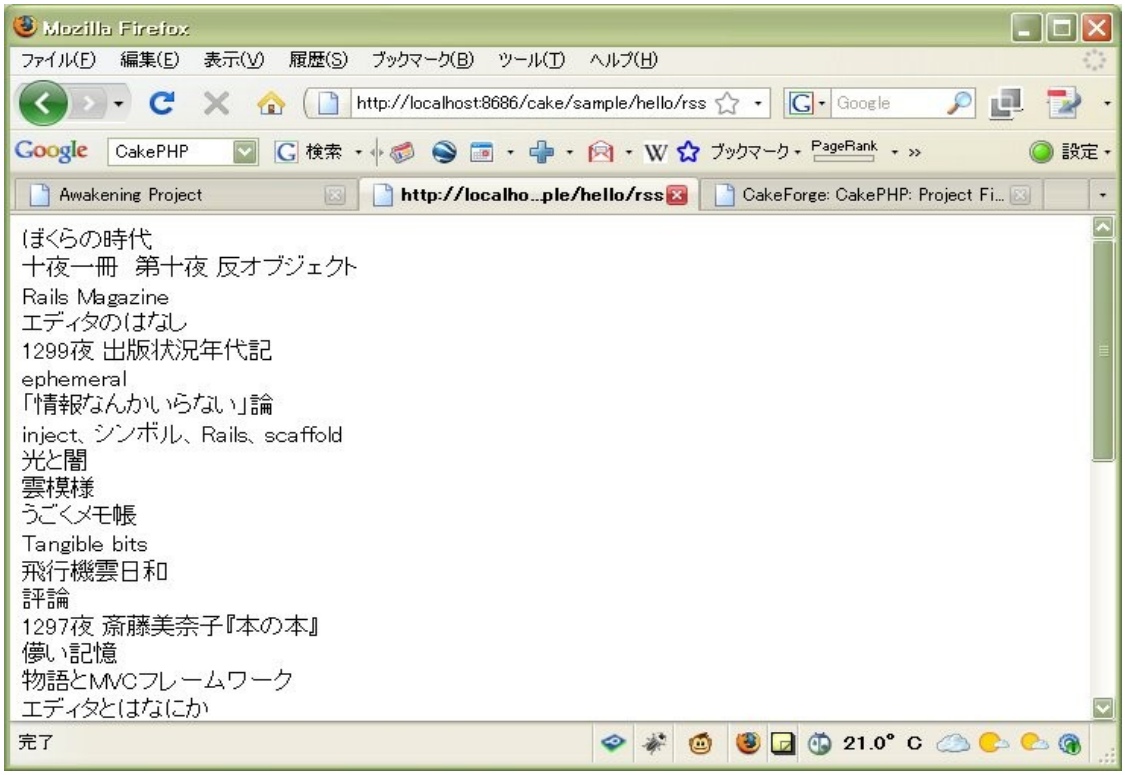


図: CakePHPのアプリケーション起動画面(http://localhost:8686/cake/sample/hello/rss)

[¥htdocs¥cake¥sample¥app¥views¥hello¥rss.ctp]]

```
<h1>サンプル見出し</h1>
<p>こんにちは! これは、CakePHPのサンプルです。</p>
<br /><br />
```

上の画面ではこのctp部分は下部に表示されるので見えていない。CtpはCake Templateの意味だろう。これをビューテンプレートと呼ぶ。この他、ヘッダーとフッターの表示をレイアウトで制御できる。上のコントローラのスクリプトのように、$this->$autoRender、$this->$autoLayoutで使用の要否を設定できる。

[¥htdocs¥cake¥sample¥app¥views¥hello¥index.ctp]]

```
<h1>サンプル見出し</h1>
<p>こんにちは! これは、CakePHPのサンプルです。</p>
<br /><br />
<?php
        echo date('Y/m/d',time());
?>
```

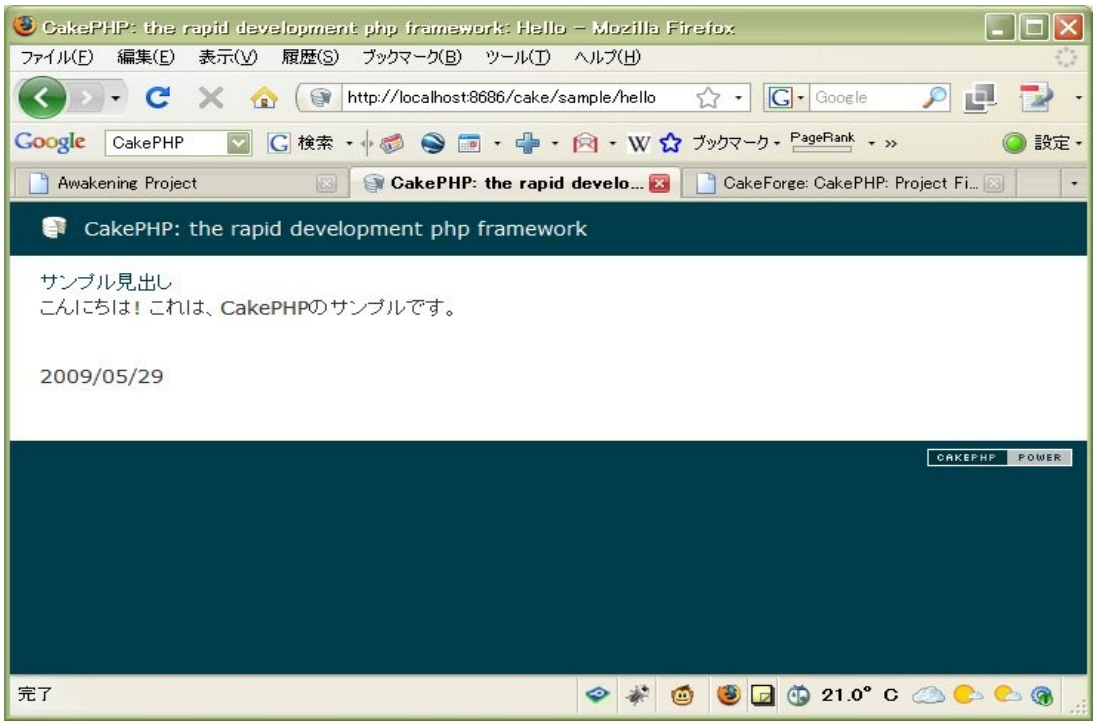


図: CakePHPのアプリケーション起動画面(http://localhost:8686/cake/sample/hello)

この表示では、$this->$autoLayouts = trueとなっているので、ヘッダーとフッターが表示されている。

[¥htdocs¥cake¥sample¥app¥views¥hello¥other.ctp]]

```
<h1>サンプル見出し</h1>
<p>こんにちは! これは、CakePHPのサンプルです。</p>
<br /><br />
<?php
        echo date('Y/m/d',time());
?>
```

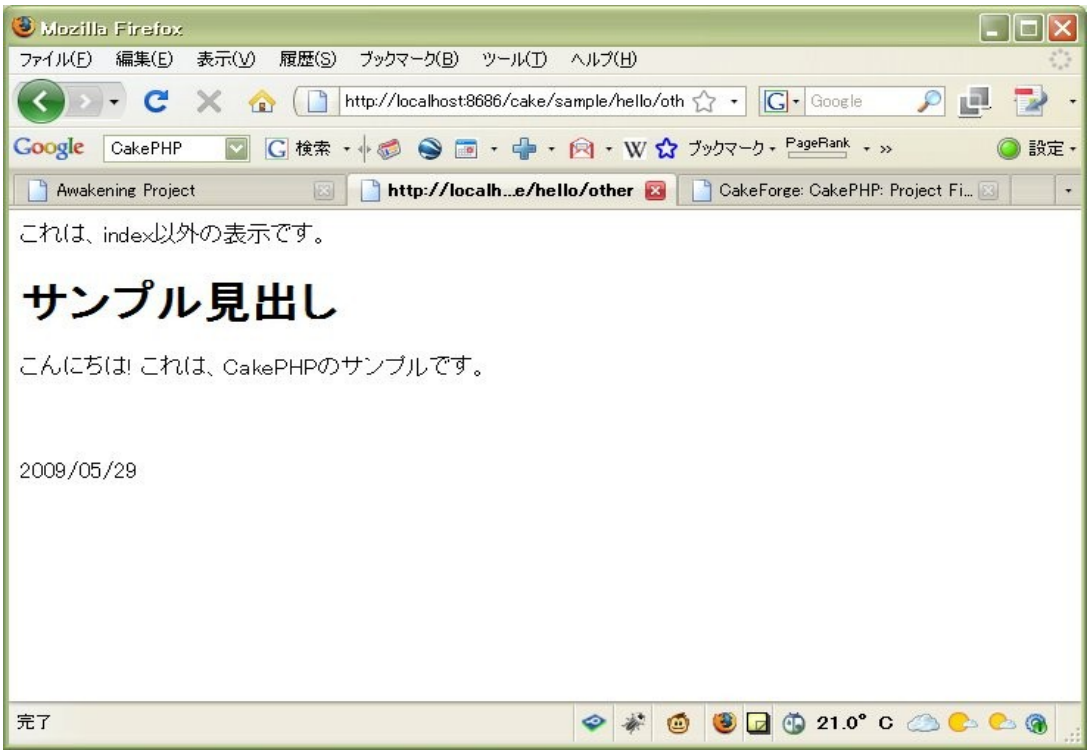


図: CakePHPのアプリケーション起動画面(http://localhost:8686/cake/sample/hello/other)

otherアクションでは、$this->autoLayouts = falseになっているので、ヘッダー、フッターは表示されない。アクション側の出力はctpに設定した出力より先に出力される。

図: CakePHPのインストールディレクトリ(¥htdocs¥cake¥sample)

インストールディレクトリのcakeにライブラリが入っている。appディレクトリがアプリケーション用のディレクトリである。

図: CakePHPのアプリケーション用デイレクトリ(¥htdocs¥cake¥sample¥app)

コントローラはcontrollersディレクトリに、ビューテンプレートはviews¥helloディレクトリに、レイアウトはviews¥layoutsに置かれる。コントローラのファイル名はRailsと同じで、コントローラ名_controller.phpのようになる。

5. Rails、Catalyst、そして、CakePHP

見掛けは似ている。しかし、インストール中心の検討では、中味までを十分に見通すことはできなかった。自分の得意な言語、Perlで書くのがやはり安心感がある。Catalystが使えそうだという感触があったのは大きな成果である。Railsは使い方がシンプルな感じがした。ディレクトリの構成がシンプルだからだろう。CakePHPはPHPのライブラリをよく知らないのが障害だった。RubyはPerlのアナロジーで走れる。PHPはそういうわけにはいかない。言語はやはり慣れが重要である。環境を含めて、勘や鼻が利かないと困るのである。

Template Toolkitの作者のAndy Wardleyは、MVC: No Silver Bullet - No Easy Route to Web Developmentと書いている(http://wardley.org/computers/web/mvc.html)。MVCは一つの方法に過ぎない。Separation of Concernsが大切と説く。誰かさんと一緒。もちろん、そんなことは当たり前だろうとも言えるし、関心を分離していく過程こそが、物事の関係性を考えることになるわけだ。しかし、現実は絡み合っていて、切り分けられることを拒絶する。絡み合いにこそ意味があるわけで、解きほぐしてしまうと意味がなくなる可能性もある。それが具体的になになのかが問題なので、抽象的な議論をしてもはじまらないわけだが・・・

ということで、いつもこのように終わらざるを得ないのだが、もう少し複雑なアプリケーションをMVCフレームワークを使う場合と別の方法を使う場合で比較するということが必要だろう。デザインの問題もあるかもしれない。テンプレートの問題もあるかもしれない。アプリケーションは一本にまとめるのがメンテナンス性がよいという意見もある。もちろん、ケース・バイ・ケースだ。その時の条件を知らねば、無意味な議論である。

次稿では、リレーショナルデータベースによるModelを含めて、MVCフレームワークについて考えてみたい。表題のポストモダン・フレームワークの意味は定かではない。コンピュータは究極のモダンを目指しているようでいて、本当にそうなのか。MVCフレームワークは物事を切り刻むモダン・フレームワークなのか。他にフレームワークはあるのか。そんなことを考えて、表題としたのだ。

(2009年5月30日)

編集後記

jscripter

巻頭言を書いて、編集後記を書くのも何か変な感じだが、誰か書いてくれないかなと、多少弱気になる。

そうそう、巻頭言に書き忘れたが、表紙の写真は4月末の旅行で新幹線から撮影した富士山である。ズームで撮った。表紙写真は最初から募集しているのだが、まったく応募がない。スクリプタに写真を望んでも無駄か^^;) 僕でも写真ぐらいはなんとかなるけどね。デジタルカメラのメディアとしての可能性はまだこれからだろう。

次号はまだMVCフレームワークネタで書こうと思っているが、フィジカルコンピューティングなぞをネタにしたいと思うこの頃。何かおもしろいネタはないかなあ・・・

ＴＳＮＥＴスクリプト通信

2009年5月30日　　2.1.000版発行

投稿規程

TSNETWiki ： 「投稿規程」のページを参照のこと

編集委員会(投稿順)

Ｙさ　　saw at mf-nokuchi2pho dot ne dot jp
機械伯爵　kikwai at livedoor dot com
海鳥　　kaityo256 at nifty dot ne dot jp
jscripter　jscripter9 at gmail dot com

著作権

使用許諾・配布条件

免責事項

編集ソフトウェア

OpenOffice.org 3.1.0 Writer

発行所(一次配布所)

TSNETWiki ： 「TSNETスクリプト通信」のページを参照のこと
