TSNET スクリプト通信

May, 2008 TSNET

「TSNET スクリプト通信」創刊の辞

五月連休に突入。ようやく「TSNET スクリプト通信」の編集に取り掛かる。昨年末、TSfree の会議室で提起した「TSNET の今後」についての議論の一つの成果として、何か会報のようなものを出そうということになった。今年の 3 月末までに投稿し、5 月連休明けに発刊の予定としていたのだ。

TSNET の母体は、NIFTY-SERVE の FGALTS の参加者である。TS はテキスト&スクリプトを意味する。FGAL(ソフトウェア・ギャラリー・フォーラム)は 1990 年代の初めに「フリーソフトウェア派宣言」(FGAL 編、翔泳社、1991 年)を出して一世を風靡した。しかし、Windows95 の登場とともに急速に発展したインターネットによって、絶頂期にあった NIFTY-SERVE のパソコン通信は衰退期に向かう。NIFTY-SERVE のパティオやチャットなどでの関係者の議論を経て、2002 年の 2 月に閑舎さんと Zazel さんの尽力で TSNET が立ち上がった。それ以来 5 年を越える年月を経て、TSNET の活動をメールの集積の中に埋もれさせておくのではなく、共同作業の形で何らかのまとまった表現として残していきたいとの思いが、「TSNET スクリプト通信」の創刊に結実した。

僕らは 1990 年代の初めから、Perl、Python、Tcl/Tk などのスクリプティング言語の発展とともに歩んできた。Ruby の誕生も見たし、FGALTS には ruby の会議室も作った。21 世紀に入って、スクリプティング言語の重要性は誰もが認めるところとなった。Larry Wall 氏が 2000 年に Perl 6 のコミュニティデザインの考え方を打ち出し、2001 年のエイプリルフールに Parrot Virtual Machine のアイデアが登場して、それはいつのまにか現実のものとなり、現在でも粘り強い開発の努力が続いている。今ではマイクロソフトでさえ IronPython や IronRuby を作っている。2007 年から CLR(Common Language Runtime)の上に DLR(Dynamic Language Runtime)を構築しつつあるのだ。Ruby は Ruby on Rails によって世界的に有名になった。わたなべひろふみさんによると英文メーリングリスト ruby-talk のメール総数は 30 万通を越えたそうだ。Python は作者の Guido van Rossum 氏が Google に所属し、Google 言語とも呼ばれている。

創刊号(1.1)は、Y ささん得意の AWK スクリプトゲーム、閑舎さんの Perl スクリプトで作る簡単 Wiki と短編小説の組合せ、機械伯爵さん力作の Python スクリプト講座と jscripter こと僕のオブジェクト指向の研究日誌の計 4 編と多彩な?内容となった。

表紙には、なんということのないグラスに冷たそうな水が入っている写真を置いてみた。文字ばかりだと寂しいなと思ったからだ。昨年の五月連休に琴平を訪れた際、丸亀の美術館の喫茶室で撮ったものである。「TSNET スクリプト通信」がこのせちがらい現実への一服の清涼剤となればとの気持ちを込めている。

このささやかな会報が TSNET メンバーの参加と協力によっていつまでも続くことを願って、創刊の辞としたい。

2008 年 5 月 3 日
jscripter

THE100

Ｙさ

1.　このゲームは

前作(MAZE3D)に引き続き、貧相な画面のゲーム第二弾(^^;)
コンピュータ３人のプレイヤーと生き残り型対戦するトランプゲームです。

元ネタは高校の頃流行ったゲームです。が、役札ルール等はでっちあげてます(さすがに記憶が...)

ところで、元ネタは、ちゃんとしたカードゲームとして売られているらしいですね、ミタコトナイケド(^^;)
大貧民は全員の順位を決められるのでオールマイティなんですけど、ｕｎｏは勝者を一人決める時、100 は敗者を一人決める時と使いわけしてました。

2. 動作環境は

一応、最近の awk なら OK だと思います。
動作確認は、
　GNU Awk 3.1.5, mawk 1.3.3 MBCS R27
で行なってます。
(...あ、全部 WinXP の DOS 窓ですね(^^;) unix 系とかで動くのかな？)

3. 遊び方は

```
>gawk -f the100.awk[ENTER]
~~~~~~~~~~~~~~~~~~~~~~~~~~
```

等で起動すると場のカードの合計とあなたの手札３枚が表示されます。

場に１枚カードを出します。
(出すカードは、下に振ってある番号を入力後、[ENTER]で確定)
場に出すと、場の合計に数字が加算されます。
場に出すと、１枚台札から手札に加わります。
以降、一人づつ順番に場に１枚づつ出していきます。
場に出すとき合計が 100 以上になった人が負けです。
誰かが場の合計を 100 以上にすると、プレイヤーにまわった回数と、そのうち何回あなたが関わったかが表示され終了します。

...これだけだと足し算の練習で終わってしまいますね(^^;)
じつはカードに役札があります。

Ａ：１または１１として加算できます。
　　前半のゲームスピードアップに使います(^^)
２：＋２または加算せずに別の人へ順番を振ることができます。
　　「指名」といふ(^^;) で、指名された人から続ける。
９：０～９任意の数で加算できます。
　　次の人にスルーできます。

１０：＋１０または－１０できます。
　　逃げの一手です（次の人が楽になる(T_T)
Ｊ：＋１１または加算せずに一人飛ばし（スキップ）できます。
Ｑ：＋１２または加算せずに逆周り（リバース）できます。
Ｋ：＋１３またはいつでも場の合計を９９にできます。
　　しょっぱなに９９にしてしまい自分が困る場合もある(^^ゞ

役札を出そうとすると、役を使うか再度問い合わせがあるので画面の指示に従って選択入力してください。
なお、ゲーム中に自分のカード番号ではなく、h か H または ？ を入力すると、役札の簡単なヘルプが表示されます。

4. 開発環境など

ソースは、ちょっと(かなり)昔に、会社の行き帰りの電車で HP100LX で改造、デバッグしたもの(C言語)を、awk に移植したものです。（C 言語版は
http://www.vector.co.jp/soft/dos/game/se031825.html にあります...）
なので awk プログラムとしては不自然な構造になっているかもしれませんが、気にしない気にしない。
あと移植して知ったのですが、gawk でそのまま C 言語の switch～case 文って使えるんですね(^^;)

5. その他

本プログラムはフリーソフトです。
著作権は作者であるＹさにあります。
ただし転載、再配布、改造、消去は自由です。
なお、（見違えるような素晴らしい）改造を行った場合は速やかに TSNet へ投稿するよう、お願いいたします。
また、このソフトを使用した事による損害が発生したとしても、損害に対しては一切の責任を負いかねます。

```
##  THE100  written by Yさ

##
##  HELP 表示
##
function helpDisp()
{
    print "";
    ##     0....5....0....5....0....5....0....5....0....5
    print "    1 ...  +1 or +11     2 ...  +2 or Nominate";
    print "    9 ...  +0  -  +9     J ... +11 or Skip    ";
    print "   10 ... +10 or -10     Q ... +12 or Reverse ";
    print "                         K ... +13 or 99      ";
}

function rnd(N){ return int(N * rand()); }  ##  乱数


BEGIN{
    ##  カード
    ## Cards[key, 52];      ##  カード本体
    split("H,D,C", StrSuit, ",");  StrSuit[0]="S";
    split(" 2, 3, 4, 5, 6, 7, 8, 9,10, J, Q, K", StrRank, ",");  StrRank[0]=" A";
    ## Table[52];           ##  台札
    ## CardPos;             ##  台札の一番上のカードを表す位置
    ## Hands[4, 3];         ##  手札

    ## プレイヤー名
    split("COM1,COM2,COM3", Player, ",");  Player[0]="YOU ";

    srand();                ##  乱数の初期化
    makeCards();            ##  カードを用意して...
    CardPos = shuffle();    ##  カードを切り...
    firstDeal();            ##  手札を配って

    ##  場に1枚出す
    ##    [0-11]の12枚は手札として配られています
    ##    場札を[CardPos-12]へ出してから[CardPos]から手札を1枚引きます
    Total=Cards["rank", Table[CardPos]] +1;  ##  0-12 -> 1-13
    Table[CardPos-12]=Table[CardPos];
    ++CardPos;

    ## Turn, Dir, Total;    ##  順番, 方向, 合計
    ## Times[2];            ##  回数
    Dir=1;    ##  YOU→COM1→...
    Turn=-1;  ##  next = Turn + Dir = 0 = YOU
    Times[0]=Times[1]=0;
```

```
    ## 0....5....0....5....0....5....0
    ## <-  10  ->&lt;-       20       ->## [D K] Total:99
    ##
    printf("¥n%*s", 10+20, " ");  ## 表示調整

    ##  場の合計が100未満の間繰り返し
    while(Total<100){
        Turn += Dir;  if(Turn>3) Turn=0; else if(Turn<0) Turn=3;
        printf("## [%s%s] total:%-2d",
          StrSuit[Cards["suit", Table[CardPos-12-1]]],
          StrRank[Cards["rank", Table[CardPos-12-1]]],
          Total);
        printf("¥n%2d >> %4s", 1+Times[0]+Times[1], Player[Turn]);
        if(Turn==0) man();
        else        com();
        if(Total<100) ++Times[((Turn==0)?(0):(1))];
    }

    exit;
}

END{
    ##  Game Over
    msg = sprintf("%4s 100", ((Total==100)?("Just"):("Over")));
    printf(" ... %-*s", 20-5, msg);
    printf("## [%1s%2s] Total:%-2d",
      StrSuit[Cards["suit", Table[CardPos-12]]],
      StrRank[Cards["rank", Table[CardPos-12]]],
      Total);
    printf("¥n¥n  %4s lose¥n", Player[Turn]);
    printf("¥n## Result: %d/%-d (%d%%)¥n",
      Times[0], Times[0]+Times[1], Times[0]*100/(Times[0]+Times[1]));
}



##
##  カード本体 と 台札 の作成
##
function makeCards(  i, j)
{
    for(i=0; i<4; ++i){
        for(j=0; j<13; ++j){
            Cards["suit", (i*13 +j)] = i;  ##  マーク
            Cards["rank", (i*13 +j)] = j;  ##  数字

            Table[i*13 +j] = i*13 +j;
```

```
        }
    }
}


##
##  カードを切る
##
function swap(p1, p2,  tmp){ tmp = Table[p1];  Table[p1] = Table[p2];  Table[p2] = tmp; }
function shuffle(  i)
{
    for(i=0; i<52; ++i)  swap(rnd(52), i);  ##  台札から適当な一枚を選んで...
                                            ##  それを i 番目のカードと入れ替えましょう
    return 0;  ##  (次の札は台札の先頭)
}


##
##  台札無いのでカードを切る
##
function reShuffle(  i, x, tmp, last, stock)
{
    ##  最初に[0-11]の 12 枚は手札として配られています
    ##  場札を[CardPos-12]へ出してから[CardPos]から手札を 1 枚引きます
    ##   → 最後の場札は[51-12]です

    last=51-12;                                             ##  最後の場札を退避
    stock=Table[last];
    for(i=0; i<=last-1; ++i)  swap(rnd(last), i);           ##  出ていた札をシャッフル
    for(i=0; i<=last-1; ++i)  Table[51-i]=Table[51-i -12];  ##  ずらして...
    Table[0]=stock;                                         ##  退避した場札を戻す
    return 12+1;  ##  (台札の先頭)
}


##
##  手札の並べ替え
##
function sortHands(who,  i, j, tmp)
{
    ##  数字順(同じ数字ならマーク順)に並べ替える
    for(i=0; i<=1; ++i){
        for(j=i+1; j<=2; ++j){
            tmp = Hands[who, i];
            if(Cards["rank", tmp]<Cards["rank", Hands[who, j]])
                continue;
            if(Cards["rank", tmp]==Cards["rank", Hands[who, j]])
                if(Cards["suit", tmp]<=Cards["suit", Hands[who, j]])
```

```
                    continue;
                Hands[who, i] = Hands[who, j];
                Hands[who, j] = tmp;
            }
        }
    }


    ##
    ##  手札を引く
    ##
    function deal(who, no)
    {
        Table[CardPos-12]=Hands[who, no];
        if(Total<100){
            ##  手札を引く
            Hands[who, no]=Table[CardPos++];
            if(CardPos>51)  CardPos = reShuffle();
        }
    }
    function firstDeal(  who, te)
    {
        for(te=0; te<3; ++te){
            for(who=0; who<4; ++who)  deal(who, te);
        }
    }


    ##
    ##  手札の表示
    ##
    function putHands(  i)
    {
        printf("¥n%2s", " ");
        for(i=0; i<3; ++i)
            printf("[%1s%2s]",
              StrSuit[Cards["suit", Hands[0, i]]], StrRank[Cards["rank", Hands[0, i]]]);
        printf("¥n%2s -1-  -2-  -3- %2s", " ", " ");
        ##        <-    2 +5*3 + 2   ->  の幅
    }


    ##
    ##  1を使う
    ##
    function use1(  tmp, no)
    {
        no = 1;  ##  仮
```

```
    if(Total+11 <= 99){
        do{
            printf("%*sHow? (1 or 11) > ", 2 +5*3 +2, " ");
            no=0;  getline no;
        }while(no!=1 && no!=11);
    }
    Total += no;
}

##
##  2 を使う
##
function use2(  tmp, no)
{
    do{
        printf("%*sNominate? (Yes:1-3 or No:0) > ", 2 +5*3 +2, " ");
        no=0;  getline no;
    }while(no<0 || 3<no);
    if(no>0)  Turn = no - Dir;
    else      Total += 2;
}

##
##  9 を使う
##
function use9(  tmp, no)
{
    if(Total<99){
        do{
            printf("%*sHow? (0-9) > ", 2 +5*3 +2, " ");
            no=0;  getline no;
        }while(no<0 || 9<no);
        Total += no;
    }
}

##
##  10 を使う
##
function use10(  tmp, no)
{
    no = -10;  ##  仮
    if(Total+10 <= 99){
        do{
            printf("%*sHow? (10 or -10) > ", 2 +5*3 +2, " ");
            no=0;  getline no;
        }while(no!=10 && no!=-10);
    }
```

```
    Total += no;
}

##
##  11 を使う
##
function use11(  tmp, no)
{
    no = 0;  ##  仮
    if(Total+11 <= 99){
        do{
            printf("%*sSkip? (Yes:0 or No:11) > ", 2 +5*3 +2, " ");
            no=0;  getline no;
        }while(no!=0 && no!=11);
    }
    if(no==0)  Turn = 2 - Dir;  ##  スキップすると COM2
    else       Total += 11;
}

##
##  12 を使う
##
function use12(  tmp, no)
{
    no = 0;  ##  仮
    if(Total+12 <= 99){
        do{
            printf("%*sReverse? (Yes:0 or No:12) > ", 2 +5*3 +2, " ");
            no=0;  getline no;
        }while(no!=0 && no!=12);
    }
    if(no==0)  Dir *= -1;
    else       Total += 12;
}

##
##  13 を使う
##
function use13(  tmp, no)
{
    no = 99;  ##  仮
    if(Total+13 <= 99){
        do{
            printf("%*s99? (Yes:99 or No:13) > ", 2 +5*3 +2, " ");
            no=0;  getline no;
        }while(no!=99 && no!=13);
    }
    if(no==99)  Total = 99;
```

```
    else        Total += 13;
}

##
##  人間main
##
function man(  tmp, no, val)
{
    sortHands(0);  ##  手札の表示
    putHands();

    do{  ##  カードの番号を入力
        printf("Which (1-3)? > ");
        no=0;  getline no;
        if(no=="h" || no=="H" || no=="?"){  ##  help !
            helpDisp();
            putHands();
        }
    }while(no<1 || 3<no);
    --no;  ## 1-3 -> 0-2

    ##  カードの役を使うかを入力
    val=Cards["rank", Hands[0, no]] +1;  ##  0-12 -> 1->13
         if(val== 1) use1();
    else if(val== 2) use2();
    else if(val== 9) use9();
    else if(val==10) use10();
    else if(val==11) use11();
    else if(val==12) use12();
    else if(val==13) use13();
    else Total += val;
    deal(0, no);  ##  手札を引く
    printf("¥n%*s", ((Total<100)?(10+20):(10)), " ");  ##  ゲーム続行/終了時表示調整
}


##
##  COM main
##
function com(  no, val, use, nextTurn)
{
    sortHands(Turn);
    ##  カス札から出す
    use = 0;  ##  役は使わない
    for(no=2; no>=0; --no){  val=Cards["rank", Hands[Turn, no]] +1;
        if(Total+val<=99 && val<9 && val!=2)
            return com_PUT_HAND(val, use, no, Turn, nextTurn);
    }
```

```
    ##  役札を出す
    nextTurn = Turn+Dir;  if(nextTurn<0 || 3<nextTurn) nextTurn=0;  ##  次は誰？
    if(Total>=100-13-1){
        if(Turn==2){        ##  COM2 は 両隣を助ける
            for(no=2; no>=0; --no){
                val=Cards["rank", Hands[Turn, no]] +1;
                if(val==11 || val==10 || val==2){
                    use=1;  return com_PUT_HAND(val, use, no, Turn, nextTurn);
                }
            }
        }else if(nextTurn!=0){  ##  次が人間ではないので助ける
            for(no=2; no>=0; --no){
                val=Cards["rank", Hands[Turn, no]] +1;
                if(val==10 || (val==12 && Turn-Dir==0) || (val==9 && Total<99)){
                    use=1;  return com_PUT_HAND(val, use, no, Turn, nextTurn);
                }
            }
        }else if(nextTurn==0){  ##  次が人間
            for(no=2; no>=0; --no){
                val=Cards["rank", Hands[Turn, no]] +1;
                if(val==1 && Total+11<=99){
                    use=1; return com_PUT_HAND(val, use, no, Turn, nextTurn);
                }
            }
            for(no=2; no>=0; --no){
                val=Cards["rank", Hands[Turn, no]] +1;
                if(val==13){
                    use=1;  return com_PUT_HAND(val, use, no, Turn, nextTurn);
                }
            }
            for(no=2; no>=0; --no){
                val=Cards["rank", Hands[Turn, no]] +1;
                if(val==10 || val==9)
                   return com_PUT_HAND(val, use, no, Turn, nextTurn);
            }
        }
    }
    ##  とりあえず自分が負けないように（役札があればそれを）出す
    for(no=2; no>=0; --no){  val=Cards["rank", Hands[Turn, no]] +1;
        if(val>=9 || val==2)  return com_PUT_HAND(val, use, no, Turn, nextTurn);
    }
    if(no<0) no=0;
    com_PUT_HAND(val, use, no, Turn, nextTurn);
}
function com_PUT_HAND(val, use, no, nowTurn, nextTurn)
{
    if(Total+val>99  || use==1){
        ##  役札なら使う
```

```
            if(val== 2){
            msg = "Nominate YOU";  Turn = -Dir;  ##  next = Turn + Dir = 0 -> YOU
            val = 0;
        }else if(val== 9){
            if(nextTurn!=0) val=0; else{ val = 99-Total;  if(val>9) val=9;  }
            msg = sprintf("+%2d", val);  Total+=val;
            val = 0;
        }else if(val==10){
            msg = "-10";  Total-=10;
            val = 0;
        }else if(val==11){
            msg = "Skip";  Turn += Dir;  if(Turn>3) Turn=0; else if(Turn<0) Turn=3;
            val = 0;
        }else if(val==12){
            msg = "Reverse";  Dir *= -1;
            val = 0;
        }else if(val==13){
            msg = "99";  Total = 99;
            val = 0;
        }else if(val== 1 && Total+11<=99){
            msg = "+11";  Total+=11;
            val = 0;
        }
    }
    Total += val;
    if(val==0)  printf(" ... %-*s", 20-5, msg);
    else if(Total<=99) printf("%*s", 20, " ");  ##  続行時表示調整
    deal(nowTurn, no);  ##  手札を引く
}
```

Wiki 風の整形ツール作成術

閑舎

ひとつ Wiki 風の整形ツールを作ってみましょう。
私が書いたショートショートがおまけ(?)です。

1. Perl でホームページ形式を生成する

Perl が C ドライブ直下にインストールされているとします。
C ドライブ直下に script というフォルダを作り、
以下の内容のスクリプトファイル ss.pl を置きます
(例えばメモ帳で ss1.txt として作成、保存し、拡張子を付けかえます。
拡張子はエクスプローラのフォルダオプションで、[表示]-[登録されている拡張子は表示しない]のチェックをはずすなど)。

```
print "<html>¥n<head>¥n<title>ホームページ</title>¥n</head>¥n<body>¥n本文¥n</body>¥n</html>¥n";
```

コマンドプロンプトを開き、

```
cd ¥script
```

として C:¥script へ移り、

```
perl ss.pl
```

とすると、画面に

```
<html>
<head>
<title>ホームページ</title>
</head>
<body>
本文
</body>
</html>
```

と出力されます。これを画面でなく ss.html に出力するには、

```
perl ss1.pl > ss.html
```

とします。こうすると、エクスプローラで ss.html をダブルクリックしたとき、実際にホームページが表示されます。

2. ホームページ出力過程を改良する

1. ではスクリプトは 1行なのに出力が数行にわたっていてその違いにとまどいます。

そこで以下のスクリプトファイル ss.pl を作ってみましょう。

```
print <<EOD;
<html>
<head>
<title>ホームページ</title>
</head>
<body>
本文
</body>
</html>
EOD
```

同様に、

```
perl ss.pl
```

とすることで出力を確かめられます。
けれども、ホームページを出力するたびにこうして何行も書くのは面倒です。
共通する部品は関数という入れ物にし、毎回それを呼び出すようにしましょう(ss.pl)。

```
print_html("ホームページ", "本文");

sub print_html() {
  my ($title, $body) = @_;
print <<EOD;
<html>
<head>
<title>$title</title>
</head>
<body>
$body
</body>
</html>
EOD
}
```

こうすると、print_html と名づけられた関数(サブルーチン)が、$title, $body という引数を受け取って、それらを適当に当てはめた数行を出力してくれます。

3. ホームページ出力過程をさらに改良する

これでは長くなってかえって不便になるだけだという読者もあるでしょう。
そこで、こうした関数集を別のファイルに入れ、スクリプト本体を簡単にしてみます(ライブラリと呼びます。もう少し要件を満たせばモジュールになります)。

```
mylib.pl
```

```
sub print_html() {
  my ($title, $body) = @_;
print <<EOD;
<html>
<head>
<title>$title</title>
</head>
<body>
$body
</body>
</html>
EOD
}

1;
```

最後の 1; を忘れないようにし、mylib.pl を C:¥script に置いてください。
そして本体 ss.pl を次のようにします。

```
push @INC, ".";
require "mylib.pl";

print_html("ホームページ", "本文");
```

こうすると、

```
perl ss.pl
```

で同じ出力が得られます。

4. 本文の行頭に * があったら見出し行として出力したい

```
sub header() {
  my ($line) = @_;
  return "<h1>$line</h1>";
}
```

が専用の関数で、呼出し側は

```
if ($_ =~ /^¥*(.*)/) {
  $lines .= header($1) . "¥n";
}
```

のようになります。

5. 本文の行頭に RIGHT: があったら右寄せで出力したい

```
sub right() {
```

```
  my ($line) = @_;
  return "<div align='right'>$line</div>";
}
```

が専用の関数で、呼出し側は

```
if ($_ =~ /^RIGHT:(.*)/) {
  $lines .= right($1) . "¥n";
}
```

のようになります。

6. 本文の行頭が ---- なら水平線を引く

```
sub horizontal() {
  my ($line) = @_;
  return "<hr>";
}
```

が専用の関数で、呼出し側は

```
if ($_ =~ /^----/) {
  $lines .= horizontal() . "¥n";
}
```

のようになります。

7. 本文が空行なら、改行を 2つ加える

```
sub doublebreak() {
  my ($line) = @_;
  return "<br><br>";
}
```

が専用の関数で、呼出し側は

```
if ($_ =~ /^$/) {
  $lines .= doublebreak() . "¥n";
}
```

のようになります。

8. 本文の行頭に &ref(...); があったら画像を出力したい

```
sub img() {
  my ($line) = @_;
  return "<img src='$line'>";
}
```

が専用の関数で、呼出し側は

```
if ($_ =~ /^¥&ref¥([^)]+)¥);/) {
  $lines .= img($1) . "¥n";
}
```

のようになります。

9. 以上をまとめます

ライブラリ mylib.pl

```
sub print_html() {
  my ($title, $body) = @_;
print <<EOD;
<html>
<head>
<title>$title</title>
</head>
<body>
$body
</body>
</html>
EOD
}

sub header() {
  my ($line) = @_;
  return "<h1>$line</h1>";
}

sub right() {
  my ($line) = @_;
  return "<div align='right'>$line</div>";
}

sub horizontal() {
  my ($line) = @_;
  return "<hr>";
}

sub doublebreak() {
  my ($line) = @_;
  return "<br><br>";
}

sub img() {
```

```
  my ($line) = @_;
  return "<img src='$line' align='right'>"; # ちょっと改変
}

1;
```

スクリプト本体
ss.pl

```
push @INC, ".";
require "mylib.pl";

$title = shift;
$lines = "";

while (<>) {
  chomp;
  if ($_ =~ /^¥*(.*)/) {
    $lines .= header($1) . "¥n";
  } elsif ($_ =~ /^RIGHT:(.*)/) {
    $lines .= right($1) . "¥n";
  } elsif ($_ =~ /^----/) {
    $lines .= horizontal() . "¥n";
  } elsif ($_ =~ /^$/) {
    $lines .= doublebreak() . "¥n";
  } elsif ($_ =~ /^¥&ref¥(([^)]+)¥);/) {
    $lines .= img($1) . "¥n";
  } else {
    $lines .= "$_¥n";
  }
}

print_html($title, $lines);
```

10. テキスト料理術

以下のショートショート原稿を C:¥script に ss.txt として保存し、コマンドラインから

```
perl ss.pl hacker ss.txt > ss.html
```

としてみてください。
準備として同じフォルダに ss.jpg という手頃なサイズの画像を置いておきます。
出力された ss.html をダブルクリックすると…。

WikiPedia などで有名な Wiki は、こうした整形ツールを改良し、機能を追加していったものです。
時間があれば、あなた自身の MyWiki を作ってみてください。

(編集者註: ss.txt は次ページ以降に掲載している。)

&ref(ss.jpg);
*幻のハッカー
閑舎 著

それは今から二年前の、初夏の休日の午後、ぼくがパソコンの前に座り、暇にまかせてページからページへとリンクをたどっている時に起こったことだ。
見たこともないページがポップアップし、また広告かと思ったぼくは、そのウィンドウを閉じようとした。
とたんに「やあ！」という大きな声がした。
ぼくはあわてて回りを見回したが、がらんとして誰もいない。

画面に目を戻すと、ページ内に四角いチャットのフォームが現れていて、何か書いてある。

「しゃべったのは私、クラウドです。ちょっと話しませんか、大介君」

ぼくはギョッとしたが、真相を知ろうと、キーボードをたたいてこう書いた。

「なぜ、ぼくの名前を知ってるんですか、ブラウザからはわからないはずなのに？」

これまで見たこともない速さで、チャット画面を文字が進み、止まった。

「それは私が君のマシンをハックしちゃったからです」

「ハックって、乗っ取りのこと？」

「そう。君の日記を読んで、パソコンを私と同じくらい愛してる人だなと思って、ちょっと話す気になったんです」

「ど、どうやって、ぼくのパソコンを乗っ取ったの？」

「説明しても難しくてわからんと思うけど。私はそこら辺のハッカー気取りの子供たちみたいにネットに落ちてるツールを使わないので」

その言葉が最後まで行かないうちに、パソコンのスピーカーからジャズが鳴り出した。

「ソニー・ロリンズの『孤独』って曲、私のお気に入りのナンバーです」

ぼくは冷汗を流しながらも、極めて平静を装って答えた。

「それって、遠隔操作でしょ。リモートデスクトップとかＶＮＣとかっていう……」

「そういう風に映るけど、私のはＯＳに登録されて動くサービスではなく、ネットワークを流れるパケット自体を加工する方法だからもっと自由なんだね」

「……」

「驚きましたか。でも、安心してください。私は悪いことはしない主義です」

「そ、そんなこと言って、信じられませんよ」

「そうですか。私は十数年ハッキングしてますが、話したのは君が最初。いわゆる何かを盗んだことは『ない』です」

「いかがわしいサイトのエッチな写真とか動画とか、盗んだことはないんですか？」

私は反撃に出てみた。

「エッチな写真ときましたか……まあ、そこにころがっていれば見ることはあるが、『君のように』集めるということはないです。そんな写真を集めるためにハックしてるわけじゃないです」

反撃どころか、かえって墓穴を掘ったかっこうだ。話題を変えないとまずい。

「じゃ、どんなハッキングをしてるんですか？」

「やはり、厳重にファイアウォールを施された組織内に侵入する計画を立て、それに成功する快感に勝るものはないですね:-)」

「どんな組織？」

「内閣府とか上場企業とか警視庁とか……」

「捕まったことはないんですか？」

「これまでのところなし」

「す、すごい。じゃ、あなたが、あの、どこにでも現れ、足跡も残さず去るという『幻のハッカー』ですね」

「どうですかね。私は長居しないし、だから足がつかないだけかもしれないし」

「あなたくらいの腕があれば、ネットワークプログラマとして相当高給が取れますよね」

「私はプログラマはやってません。一般のパソコントラブルのサポートをやっております。お客さんからは毎日どなられながら仕事してますが、楽しいです」

「ソフトじゃなく機械自体の故障なら直らないし、文句いうのは客のほうが悪い！　でも、サポートの仕事って安くて割に合わないんじゃ？」

「収入は別にいらないのです。企業が買収を発表する前など、組織内に流れる情報が傍受できると株を買って売り抜け、数百万くらい、わけなく入ってきます。これは盗んだわけじゃない、たまたま耳に入ってきただけです」

「うーん、うらやましい。でも、侵入先で何かしゃべりたくなることはないですか？」

「時々、家庭内のトラブルで悩んでいる人の日記を見たりすると、アドバイスしたくなることもあるけれど、私自身失敗者なので、しても意味がなく、放ってます」

「あなたのしゃべり方を聞くと穏やかで、某チャンネルの掲示板にひしめく自称ハッカーたちとはだいぶ違いますね」

「私、某チャンネルは好きになれないですね。厨房とかってせりふも……ゲームしてるほうがよほどまし」

「ところで、なぜ、あなたはぼくのような者と話したくなったんですか？」

「それは、きみが持ってた一枚の写真ファイルのせいです」

「写真って、どれですか？」

「マイドキュメントフォルダにあった mother.jpg っていう写真のことです」

「ああ、亡くなった母の写真ですか」

この時、クラウド氏の答が返ってくるまでに少々間があった。

「亡くなったって、いつ頃ですか？」

「かれこれ、四五年前になります。それが何か？」

「……彼女は私にとっても大切な人でした。十数年前、私のようなパソコンおたくとは一緒に住めないと、男の子を連れて出ていってしまった」

「えっ、母さんと一緒に住んでいた人……それじゃ、あなたは！」

「もうチャットはやめましょう。私のハッカーとしての知性がこれ以上ここにいるのは危険だと警告してます。私の最初で最後のチャットの相手をしてくれた大介君、ありがとう……そしてさ、よ、う、な、ら」

その瞬間、ぼくのパソコンは急にシャットダウンした。
あわてて再起動し、ブラウザの履歴をチェックしてみたが、どこにも直前にアクセスしたページはなく、チャットのログも勿論残ってはいなかった。

それから一ヶ月、ぼくはほとんど毎日のように、あの見たこともないページのポップアップと「やあ！」という声を心待ちにしたが、それは二度と起こらなかった。
あれは幻のハッカーという幻だったのだろうか。

専門学校を卒業したぼくは今、パソコンサポートスタッフの仕事をし、お客さんや上司に怒られながら、懸命にパソコンという不機嫌な機械と格闘している。

RIGHT:完

Python で学ぶビット演算(Python 3000 対応)

機械伯爵

0. はじめに

0-0. 高等学校情報科の論理演算の扱い

高等学校指導要領によれば、普通教科「情報」の３つの科目「情報Ａ」「情報Ｂ」「情報Ｃ」のうち、プログラミング言語の扱いをはっきりと示しているのは、「情報Ｂ」の科目です。
特に「コンピュータの仕組みと働き」という項目に於いては、簡単なビット演算の理論についても触れられています。
この中で「プログラミング言語の習得が目的とならないよう」と釘をさされているように、あくまでプログラミング言語は情報処理の「手段」であり、目的ではありません。
習得が困難な言語を学習に用いると、まさにその懸念が現実のものとなるでしょう。
かといって、教育専用のプログラミング言語を用いるのがいいかといえば、「その知識が将来的に活かせる」という点で、やや疑問が残ります。
無論、教育専用言語を用いた学習が全くムダになることは無いでしょうが、その言語特有のルールが将来的にほとんどつかえないとなれば、非効率的であることを否めません。
そこで、数あるスクリプト言語の中でも、習得が特に容易で、かつ実用的なプログラミング言語として有名な Python を用いて、ビット演算の学習を行うためのポイントを紹介したいと思います。

0-1. プログラミング言語 Python

Python は、Google で採用されたことで一躍有名となったプログラミング言語ですが、徹底的にオーソドックスなスタイルを追求した文法構造であるため、プログラミング初学者や非プログラマの為の手軽なスクリプト言語、あるいはプロトタイピング用の実行可能仮想コードとしても高い評価を得ています。
Python は、テキストファイルにスクリプトを書いてそれを実行することも勿論可能ですが、簡単な動作実験であれば、コマンドラインから呼び出して、１行づつスクリプトを打ち込み、評価するコマンドラインモードが便利です。
このコマンドラインモードは、「インタープリタ」や REPL(read-evaluate-print-loop)などとも呼ばれ、人工知能開発言語として有名な Lisp などで採用され、手軽な実行環境として重宝されています。
プログラミング言語によっては、このようなコマンドラインでの対話式では、機能が制限される場合もありますが、Python ではそのような制限は殆どありません。
そのわずかな制限の殆どは、コマンドラインコンソール自体の制限に由来するものであり、Python の言語的な使用に由来するものは一切無いと言って良いでしょう。
Python 言語自身の入門については、ネットの上を検索すればそれこそ山ほど見つかりますので、ここでは省略させて戴きます。
本文書は Python の次期主力バージョンである ver.3.0 以降を想定して説明します（Python ver.3.0 の情報については、Python 3000、もしくは py3k でも検索できます）
Python 2.x 以前を対象としたテキストを使用しても Python の勉強はできますが、本文書のスクリプトは Python 3.0 以降でしか動作しない機能や書式を用いていますので、気をつけてください。

1.学習範囲

1-0.学習範囲の設定

ビット演算の領域を高等学校の情報の授業で扱おうとすると、様々な問題がその前に横たわっているのに気づくでしょう。
意義、実用性の問題とあいまって、最大の問題は「どこまで学習の対象とするか」という、学習範囲の問題です。
コンピュータの黎明期、一般人がコンピュータに触れる機会など殆ど無かった時代に情報処理の話をしようとすれば、自ずとコンピュータの仕組みやビット演算の仕組みを話す必要がありました。
現在、実習用としてコンピュータが一人一台使える時代に、同様な話をしても、使いようの無い知識として受け取られる危険性があります。
昨今の風潮では、ビット演算／論理演算は、ハードウェアを直接弄る職人プログラマの領域だと考えられている節もあります。
実際には職人どころか、一般の非プログラマの目の前にすら、生のビット演算光景が観察できます。
例えば、IPアドレスのビットマスクなど、簡単なビット演算以外の何物でもありません。
その他、アイディア次第では、様々な場面で二進法演算は役立ちます（もちろん、パズルのような頭の運動にもなります)
そのためにも、折角覚えても使いようのない知識はできるだけ省いて、範囲を組み立てる必要があるでしょう。
そこで私が提案する範囲設定は、「ビット演算の中で、非プログラマでも実用可能な知識」としたいと思います。
具体的な領域については、以下にお話していきたいと思います。

1-1.二進法(binary)の領域

通常、ビット計算や論理演算は、二進法の知識の説明から始まります。
二進法は、ビットを数値化する上でなくてはならない知識です。
しかし、それとセットでかならずついてくる「二進法計算」の必要性については、かなり疑問です。
そもそも、コンピュータに計算させるための手段として二進法を使うのに、その二進法計算を人間が手計算でやっては本末転倒です。
十進法⇒二進法の変換、そして二進法⇒十六進法の変換は、データの基礎知識として必要でしょう。
しかし、十進法⇒十六進法や十進法⇒十六進法の変換は、話としてはともかく、覚えこませる必然性は見当たりません。
底の変換は、十進法⇒二進法⇒十六進法までができれば、あとの計算はコンピュータにお任せしましょう。

1-2.数値の表現（実装）法

コンピュータのビットに関する知識といえば、最初に０と自然数の実装方法を学び、その次には２の補数による負の整数の実装法、さらには浮動小数点数の実装方法に進みます（ベテランの先生であれば、ゾーン形式やパック形式の数値表現などの知識も、披露してくれるかもしれません）
特に数学系／工学系の情報科の先生としては、このあたりは非常に「面白い」分野であり、ついつい熱弁を振るってしまうのも無理は無いのですが（知識を熱く語らない先生なんて、魅力半減ですからね）しかしここに大きな落とし穴があります。
二進法は基本概念、２の補数や浮動小数点数の話は実装の問題、と、両者の意味合いが微妙に異な

るので、両方を続けて話されると、生徒が少なからず混乱します。
当然ですが、ビットによる負の数や浮動小数点数の実装は、完全に二進法のルールには従ってはいません。
二進法の概念を完全に理解しないうちに、負の数や浮動小数点数の話に進めるのは危険です。
よって私は、「０と自然数」の実装方法だけに限定することをお勧めします。

1-3.論理演算／ビット演算

情報の授業で扱われる論理演算に関係するものとしては、、論理和（OR ∨）、論理積（AND ∧）、論理否定（NOT ¬）の他に、排他的論理和(XOR,NOR)の話があります。
このうち、論理和、論理積、論理否定は基礎ですので、勿論話すべきでしょう。
排他的論理和（「どちらか一つ」）については、は、論理演算としてはやや高度な部類に入りますが、概念的にはわかりやすいですし、ビット演算としても実用としても使いやすいので、これは範囲に入れるべきでしょう。
ただし、排他的論理和の延長としての、半加算器や加算器の話は、やや複雑な話になるので、割愛したほうが良いでしょう。
さらに、ビット演算のシフト演算は、掛け算／割り算の原理などで面白い分野ではあるのですが、その話に付随する桁あふれ問題などを考えると、これも範囲に入れるべきではないでしょう。

1-4.記憶容量

ビット／バイトの話から、８ビットまでの二進法は、勿論問題にすべきです。
バイトに対し、８ビットであることを正確に示す「オクテット(octet 「八重奏」などという意味でも用いる)」という言葉を使ってもいいのですが、まぁ、今現在では殆ど意味が無いでしょう。
基本的にデータはバイト単位で処理しますので、８ビット以上の記憶容量の話については、あまり触れる必要は無いと思われます。

1-5.その他

発展的な内容については、とりあえず基礎が理解されればいずれできます。
まずは欲張らずに、基礎的な部分を固めることから進めましょう。
せっかくパソコンが使えるのですから、紙上での計算より、コンピュータを立ち上げて、そこで「実験」させることによって、生徒の感性に訴える方法のほうが、早く深い理解につながるのではないかと思います。

2.授業としてPythonで使えるツール群

2-0.Pythonのコマンドライン・コンソール

PythonをWindowsにインストールすると、スタートメニューから「Python(command line)」が選べますので、これでスタートできます。
Pythonが既にインストールされている Linux や Mac OSX のような UNIX 系の OS の場合、適当なシェルを開いてそこから'python'と打ち込めば、コマンドラインモードでPythonが立ち上がります。
コマンドラインを終了させるには、exit()関数及びquit()関数が使えます。
なお、立ち上げればわかると思いますが、行頭の'>>> 'はプロンプトですので、打ち込む必要はあ

りません。

※プロンプト(prompt)：コマンドラインで、入力可能な行の行頭に現れるマーク。 DOS / Windows 系コマンドプロンプトでは'>'、UNIX 系シェルでは'$'が、プロンプトとしてよく使われる

2-1.二進法リテラル

※リテラル(literal)：数値や文字を直接プログラムの中に書き込むための書式

Python(ver.3.0以降)では、以下のように二進法を直接書くことができます。

```
>>> 0b100
4
>>>
```

二進法表記の前に、'0b'の文字をつければ、二進法リテラルと判定されますので、二進法として数値処理されます。
リテラルですから、数値としてそのまま計算もできます。

```
>>> 0b100 + 0b011
7
>>>
```

0b100が４、0b011(桁を揃えてみたが、もちろん0b11でも良い)が３ですので、答えが７となります。
なお、十六進リテラルは、頭に'0x'をつけます。

```
>>> 0x100
256
>>>
```

2-2.十進法数を二進法で表示する。

十進法の数値を二進法表示するには、bin 関数を使用します。

```
>>> bin(100)
'0b1100100'
>>>
```

同様に、十六進法表示ならば、hex 関数を利用します。

```
>>> hex(100)
'0x64'
>>>
```

2-3.論理演算子

ビット演算で論理積(合接 ∧)を求める場合は、&演算子を利用します。

```
>>> 0b111000 & 0b001100
8
>>> bin(0b111000 & 0b001100)
'0b1000'
>>>
```

論理和(離接 ∨)は|演算子を利用します。

```
>>> 0b111000 | 0b001100
60
>>> bin(0b111000 | 0b001100)
'0b111100'
>>>
```

論理否定（¬　ビット反転）は～演算子を利用しますが、通常に使用すると、多分予想とは異なる結果が出ます。

```
>>> ~0b1010
-11
>>> bin(~0b1010)
'-0b1011'
>>>
```

これは２の補数を負の数として利用しているために起こる現象ですので、反転の結果を正確に表示しようとするなら、その結果に対し、表示ビット分のビットマスクとの論理積を取る必要があります。
たとえば４桁分のビットを表示させるには、0b1111 との論理積を取ります。

```
>>> ~0b1010 & 0b1111
5
>>> bin(~0b1010 & 0b1111)
'0b101'
>>>
```

表示に関して、上記のようなわずらわしさを省略するために、以下のような関数を作成しておくと便利です。

```
# 表示ビットが可変（デフォルトは８桁）
bits = lambda x, y = 8: print(('0' * y + bin(x & 2 ** y - 1)[2:])[-y:])
```

lambda 文になじみの無い方は、以下の定義が等価となります。

```
def bits(x, y = 8):       # デフォルトは8bit
  x = x & (2 ** y - 1)    # マスク
  x = bin(x)              # 二進法表示
  x = x[2:]               # '0b'を除去
```

```
  x = '0000000' + x       # 0を表示
  x = x[-y:]              # 表示桁数を表示
  print(x)
```

このbits関数はモジュールに収納し、実習開始直後にインポートして使用可能にしておくと良いでしょう。
bits関数を実際に使用した場合、以下のように表示されます。

```
>>> bits(~0b10101010)
01010101
>>>
```

排他的論理和は、^演算子を使用します。

```
>>> bits(0b10101010 ^ 0b11110000)
01011010
>>>
```

排他的論理和(XOR,NOR) A ^ Bは、(A | B)&(~(A & B))に等価です。

A XOR B = (A OR B)AND(NOT(A AND B))
= (A ∨ B)∧(¬(A ∧ B))

2-4.論理包含

論理包含(AならばB, A ⇒ B)は、条件付き評価式を用いれば書けますが、かならず「～でないならば(¬A ⇒ C)」のelseがくっつくことになります。
即ち、「ＡならばＢ、そうでないならＣ」とし、B if A else Cと書きます。

```
>>> 100 if True else 200
100
>>> 100 if False else 200
200
>>>
```

2-4.連続バイトデータオブジェクト(bytes, bytearray)
連続したバイトデータを扱うツールとしては、bytesオブジェクトやbytearrayオブジェクトが便利です。
たとえば、'abcde'という文字列を、文字コードの数値として扱いたい場合は、以下のように書きます。

```
>>> b = b'abcd'
>>> b[0]
97
>>>
```

bytesオブジェクトのリテラル表記では、文字列の前にbと書きます。

bytes オブジェクトを数値コレクションより生成するには、以下のようにおこないます。

```
>>> b = bytes((97,98,99,100,101))
>>> b
b'abcde'
>>>
```

※コレクション(collection):複数のオブジェクトを収納するコンテナオブジェクト。タプル、リスト、辞書など

bytes データは、全体としては文字列として表示されます（ASCII データの範囲を超えると、エスケープ文字を使って表示されます)
bytes データは不変オブジェクト（内容を変更できない）ですので、可変オブジェクトとして扱うには、bytearray オブジェクトを用います。
bytearray オブジェクトにはリテラルは無いので、bytes データなどから生成します。

```
>>> ba = bytearray(b'abcde')
>>> ba
bytearray(b'abcde')
>>>
```

無論、数値コレクションからも、変換可能です。

```
>>> ba = bytearray((97,98,99,100,101))
>>> ba
bytearray(b'abcde')
>>>
```

bytearray オブジェクトは可変オブジェクトなので、内容の差し替えが可能です。

```
>>> ba
bytearray(b'abcde')
>>> ba[0] = 120
>>> ba
bytearray(b'xbcde')
>>>
```

bytearray オブジェクトを bytes オブジェクトに戻すには、単純に bytes を関数として呼び出して新たにオブジェクトを生成すれば可能です。

```
>>> ba
bytearray(b'xbcde')
>>> bytes(ba)
b'xbcde'
>>>
```

当然ですが、bytes オブジェクトや bytearray オブジェクトの１アイテムの情報量は１バイトなので、１アイテムにつき 0～255 の数値しか収納できません。

ファイルをバイトデータとして読み込む場合は、open 関数のオプションに'rb'を付けます。

```
>>> f = open('op.txt','rb')
>>> k = f.read()
>>> k
b'Hello! Python World!!¥r¥n'
>>>
```

バイトデータを一気に数値表記する方法は意外と簡単で、tuple や list に変換してやればいいのです。

```
>>> name = b'Yasuo Fukuda'
>>> name
b'Yasuo Fukuda'
>>> tuple(name)
(89, 97, 115, 117, 111, 32, 70, 117, 107, 117, 100, 97)
>>>
```

以上がビット演算／論理演算の学習に利用できそうなツールですが、この他の Python の一般的なツール（例えばリストや辞書、集合、タプルなどのコレクション、ファイルなど）も勿論十分に活用して下さい。

3.実際の実習

3-0.「実験」的に実習を行う

簡単とはいえ、プログラミング言語まで持ち出して実習を行うわけですから、普段のワープロや表計算ソフトを活用した実習と同様のことをしていては面白くありません。
Python のコマンドラインインタープリタを用いる以上は、決められた通りにキーを押すだけでなく、生徒が色々自分で試してみることを、是非推奨してください。
色々試した結果どうなったのか、あるいは、こうしたらどうなるか予想して、実際やってみてどうだったなどの実験的要素が、授業をよりエキサイティングに変えてくれるはずです。

3-1.まずは「計算機的」に使う

まずは、誰でも使える「四則演算機」としての使い方。
＋やーの記号はそのまま使えますし、×は＊、÷は／を用います。
注意すべきは小数点を含む浮動小数点数で、割り算を行うと、整数同士の割り切れる割り算であってもかならずこの浮動小数点数型となって表示されます。
これが、0.5 や 0.25 などの２の累乗のものなら問題無いのですが、１÷５などの結果として出てくる 0.2 などは、やや意外な表示を行います。

```
>>> 1 / 5
0.20000000000000001
>>>
```

これは、浮動小数点数が二進法表現を使用しているために起こる現象です。
簡単に言えば、0.2 などの小数は二進法では正確には表せないのです。
この問題は、見かけ上については、表示関数である print を使えば解決します。

```
>>> print(1 / 5)
0.2
>>>
```

このように、小数表示にやや難があるものの、電卓では真似できない技もあります。
たとえば累乗（べき乗）計算は、'＊'を二つ並べた'＊＊'を用います。
１００の１００乗など、電卓ではとても表示できません（２０１桁ですので）が、Python インタープリタなら楽勝です。

```
>>> 100 ** 100
10000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
>>>
```

また、演算順序は通常の計算と同じ（乗算・除算が先で加算・減算が後）ですし、カッコも使えますので、慣れれば電卓よりはるかに便利です。
まずはこのスーパー計算機に慣れるまで、生徒に存分に弄らせてみてください。

3-2. 二進法の計算

さて、計算機に慣れたら、次は二進法です。
といっても、まずは二進法リテラルを打ち込ませて、そのまま十進法表示される数値を確認するだけです。
このとき、特に 0b1, 0b10, 0b100, 0b1000 など、キリの良い数値を必ず確認させてください。

```
>>> 0b1000
8
>>>
```

もし、教室で事前学習を行う機会があるのなら、1, 2, 4, 8, 16, 32, 64, 128 と書いた８枚のカードを組み合わせれば、0～255 までの数値が表現できることを確認させておくと良いでしょう（この授業中に黒板／ホワイトボード／プロジェクタなどで説明しても良い）

＜授業例＞
1 のカードだけなら、1 か 0 しか作れない(0 はカードを使わないことで作る)
1 と 2 のカードなら、0～3 の数字が作れる。
1 と 2 と 4 のカードなら、0～3 の数字他に、4 を足して 4～7 の数字も作れる。
1 と 2 と 4 と 8 のカードなら、0～7 の数字他に、8 を足して 8～15 の数字も作れる。
1 と 2 と 4 と 8 と 16 のカードなら、0～15 の数字他に、16 を足して 16～31 の数字も作れる。
1 と 2 と 4 と 8 と 16 と 32 のカードなら、0～31 の数字他に、32 を足して 32～63 の数字も作れる。
1 と 2 と 4 と 8 と 16 と 32 と 64 のカードなら、0～63 の数字他に、64 を足して 64～127 の数字も作れる。

1と2と4と8と16と32と64と128のカードなら、0～127の数字他に、128を足して128～255の数字も作れる。
つまり、1,2,4,8,16,32,64,128のカードを組み合わせれば、0～255の数字が作れる。

これを、それぞれ桁数と考え、たとえば十進法の 101 が 100 + 1 であるように、二進法の 0b101 も 0b100 + 0b1 であることを説明します。
そうすればイメージとして、二進法を理解しやすくなります。
あとは 0b1 + 0b1 = 0b10 の桁上がりの原理さえ理解できれば簡単なのですが、これも実際の十進法の数値を見つつ話せば、かなり分りやすくなります。
なにより計算が嫌いな生徒でも、計算はコンピュータが代わってやってくれるので、楽勝です。

同様に、十六進法リテラルやhex関数を用いて、十六進法の感覚を掴みます。
特に二進法リテラルを十六進法表記に、十六進法リテラルを二進法表記に変換すると、どうして十六進法が便利なのかがわかります。

3-3.論理演算実習

論理積、論理和、論理否定、排他的論理和の実際の計算は、先ほど作ったbits関数を使用したほうが理解しやすいと思われます。
'binary.py'というモジュールファイルにbits関数を登録した場合、とりあえずインポートさせます。

```
>>> from binary import *
>>> bits(255)
11111111
>>>
```

Pythonのモジュールは面倒な手間は一切不要で、行いたい処理や、登録したい関数やクラスの定義を書いたファイルを用意するだけです（ファイル名の拡張子は.pyとします）から、非常に簡単です。
Pythonのパスの通っているところ(Pythonフォルダの中のLibフォルダの中など)に入れておくと良いでしょう。
論理演算に関しては、実際にいろいろ試すのが一番です。
特に、bits関数は、十進法表記同士の計算結果も二進法として表示しますので、十進法で論理演算問題を生徒に予想させ、実際にコンピュータに計算させた結果と見合わせるというのも面白いでしょう。

3-4.論理演算実習の応用例

ここで、ビット演算を用いた例を挙げてみます。

「１２人のグループに０～１１番の番号をつけて出欠席を管理します。出欠席状況を簡単な１６進コードで表し、欠席管理や特定メンバーの出席状況を示すには、どのようにすればよいでしょうか？」

解答の方法はいくつかあるでしょうが、簡単な例としては以下の通り。

「管理する数値をＸとし、全員出席の場合の数値を２の１２乗から１を引いた数とする（この数値は、

二進法で表示すると、ちょうど１を１２個並べた数値となる）
　出欠コードを作成するには、「２の欠席メンバーのナンバー乗」を合計したものと全員出席コードの排他的論理和を取る。
　出欠コードは十六進法で管理する。
　出欠コードから、特定メンバーの出席を確認するには、出欠コードと「２の欠席メンバー乗」との論理積を取り、０かどうかを確認する」

```
>>> ALL = 2 ** 12 - 1 # 全員出席コード
>>> bin(ALL)
'0b111111111111'
>>> DATA = ALL ^ (2 ** 3 + 2 ** 5 + 2 ** 7) #3,5,7番が欠席
>>> hex(DATA) # 管理データコード(3桁の十六進コード)
'0xf57'
>>> DATA & 2 ** 4 # 4番の出欠確認(0でなければ出席)
16
>>> DATA & 2 ** 5 # 5番の出欠確認(0でなければ出席)
0
>>>
```

　排他的論理和を取れば、特定のビットを簡単にOFFにできます。
　また、論理積を用いれば、特定ビットの状況を簡単に調査できます。
　表示させると０か０でないか、という微妙な表示方法になりますが、プログラムの中で使用すれば、０か０でないかは、条件式の中では「偽か真か」として用いることが出来ます。
　なお、参考までに、プログラムとして出欠オブジェクトを作った場合どのようになるかの動作コードを、載せておきます。

```
>>> class Attendance:
...     def __init__(self, number, *absence):
...         self.ALL = 2 ** number - 1
...         self.PRESENCE = self.ALL
...         if absence:
...             self.absence(*absence)
...     def code(self):
...         return hex(self.PRESENCE)[2:]
...     def presence(self, number):
...         if self.PRESENCE & 2 ** (number - 1):
...             print('No.%d is presence.' % number)
...         else:
...             print('No.%d is absence.' % number)
...     def absence(self, *number):
...         for n in number:
...             self.PRESENCE ^= 2 ** (n - 1)
...
>>> data = Attendance(100,88,55)
>>> data.presence(77)
No.77 is presence.
>>> data.presence(88)
No.88 is absence.
```

```
>>> data.presence(55)
No.55 is absence.
>>> data.absence(77)
>>> data.presence(77)
No.77 is absence.
>>> data.code()
'fff7feffffbfffffffffffff'
>>>
```

4.おわりに

ビット演算や論理演算、文字コードの話は、シンプルな部分を取り出し、さらにスクリプト言語で実際に動かすことによって、学習の難易度が格段に易しくなります。

紙の上では理解しづらい概念も、実際に計算式を入れてみれば一発です。

プログラミング言語とビット演算、どちらも「難しい」と敬遠される領域ですが、両者をあわせて実習すれば、生徒の興味を十分引き、楽しく学習することが可能です。

どうぞ、「コンピュータ本来の醍醐味」を、ビットを叩くことによって、生徒に体験させてください。

オブジェクトを廻る日記

jscripter

Foreword

オブジェクト指向プログラミングという用語を最初に聞いたのは、おそらくCマガでC++を知ったときか、Javaが登場した時(アーサー・バン・ホフ他著、「かんたんJAVA」、アジソン・ウェスレイ・パブリッシャーズ・ジャパン、1996年)だと思うが、定かではない。ビル・ゲイツがWindowsNTを開発する時にC++を使うように指示して大混乱に陥った話を読んだ時も、へーそんなものなのかと漠と思った程度の認識であった(G・パスカル・ザカリー著、「闘うプログラマー 上下」、日経BP出版センター、1994年)。

そんな具合だから、僕がPerlを知って以来、オブジェクト指向を実際に使ってみるまでに10年を要し、オブジェクト指向を極めてみようと考えるまでにさらに5年を要したことになる。本稿は、オブジェクト指向の概念を極めようとして辿った道筋の記録、著者のWeb日記「日曜プログラマのひとりごと」に掲載している関連記事を時系列順に集成したものである。

そのきっかけの最大のものは、昨年(2007年)のLarry Wallのたまねぎの状態11のトークだった。Perl 6の話の中にMooseというPerl 5用のモジュールが紹介されていたのだ。Mooseのドキュメントの引用文献を読み、孫引きを始めるとオブジェクト指向とコンピューティングの歴史との関わりが見え始めた。

結局、オブジェクト指向を調べていくと、フレーム理論、意味ネットワーク、グラフ理論、指示的意味論(denotational semantics)、集合論までを辿ることになった。

僕は、いささか単純ではあるが、ロラン・バルトのERCやコノテーション(内包)とデノテーション(明示)の入れ子構造の議論(ロラン・バルト著、「零度のエクリチュール　付・記号学の原理」、みすず書房、1971年)によってオブジェクト指向が腑に落ちた。ERCはExpression-Relation-Contentの言葉(記号)の持つ二重性を示す。デノテーション(denotation)はER(ERC)という言葉の入れ子構造を表す。C(Content)にERCを組み込み、内容をさらに説明する構造である。言葉(記号)は自然とそのような性質を示すのだから、プログラミング言語も当然、その方向に進むべきではないかということである。入れ子になったERCはオブジェクト指向の概念ではオブジェクトの属性と考えることができる。

調べたものの中に、フレーム理論などの知識表現の理論の一つとして”Unit Package”(Stefik, 1979)があった。現在のオブジェクト指向における様々な概念について、豊かなインスピレーションを与えてくれる。オブジェクトは、より大きな意味ネットワークの仕切られた集合(Units)であり、Nodes、LinksとAttached proceduresからなる構造を持ったものとして考えることもできるだろう。Unit Packageでは、Nodes間の構造を定義するLinksはいくつかの”aspects”を持つslotとして生成され、slotの持つ継承(INHERITANCE ROLE)について4種類のモードがある。Unit PackageはMOLGEN(Stefik,1981)という分子遺伝学の実験計画法のために開発されたものである。

(2008-05-05)

Experimental

10/18/2007 (Thu.)

[サイバースペースカウボーイズ] オブジェクト指向

第百六十七回 オブジェクトオーバードライブ 前編ネタ。TSNET でも話題になっているが・・・

僕も「オブジェクト指向 Perl モジュールミニ入門」を「実践実用 Perl」に書いたので、少しは勉強したが、オブジェクト指向についてそれほど理 解できているとは言いがたい。実際上、オブジェクトを取り扱うのは、オブジェクト指向モジュールを使う場合に限られるからだ。人の書いたものを使う場合に は、使い方だけの知識になってしまってオブジェクトについて理解する必要はほとんどない。モジュールを作る側にまわらないと理解する機会はそれほどないということになる。

最近、chromatic、Damian Conway、Curtis "Ovid" Poe 著、株式会社ロングテール、長尾高弘訳、「PERL HACKS」(オライリー・ジャパン、2007 年)をつい買ってしまったが、なかなかワクワクするクレジットやまえがきがあったからだ。第 5 章「オブジェクトのハック」のまえがきから引用してみよう。

> Perl のことを知れば知るほど、高水準の抽象を作り、利用するための選択肢は増えていく。次に同僚たちが解決できない困難な問題にぶつかったら、 自家薬籠中の物としている OO トリックを取り出し、にっこり笑ってこう言おう。「だいじょうぶだよ。Perl は何でもできる。」

もちろん、Perl にはカプセル化、継承、演算子のオーバーロードそしてポリモーフィズムだろうが、総称だろうが、多重ディスパッチだろうが、永続オブジェクトだろうがすべてある。

・・・ということで、少し勉強してみようかという気持ちになってきた^^)v

10/20/2007 (Sat.)

[プログラミング] オブジェクト指向プログラミング - 裏返しオブジェクト?!

Perl の場合は、パッケージがクラスで、リファレンスが指し示すものがオブジェクトであり、サブルーチンがメソッドになる。ハッシュのリファレン スは祝福(bless)されてパッケージに結び付けられる。「PERL HACKS」の「オブジェクトをハックする」を読んでいると、「裏返しオブジェクト」の実装が出てくる。特に Class::Std モジュールを使うと便利らしい。「裏返し」は「inside-out」の訳である。Class::Std モジュールのドキュメントには「Perl Best Practices」(2005 年)の該当部分が転載されている。ハッシュベースのオブジェクトから「裏返し」オブジェクトへと、Perl のオブジェクト 指向も進化しているのか・・・

- オブジェクト指向プログラミング - Wikipedia
- Object-oriented programming - Wikipedia, the free encyclopedia

問題は、オブジェクト指向プログラミングに適した題材・素材を持っているかどうかということである。単にフィルターを書くだけならオブジェクト指向は不要である。ここ数年間で、本サイトでは、日記記事から年間のカテゴリ別インデックスや月ごとのインデックスを出力するだけでなく、月単位で RSS/Atom に変換して出力したり、Timeline データ用の XML を出力するようになった。これらはすべてフィルターである。最近では、 CodeZine 記事の連載を駆動力にして、Atom データを元に SQL データベースを構築するようになっている。オブジェクト指向プログラミングを、日記記事データの構造をオブジェクト化して、ビュー(View)をコントロールするようなシステムへの転換に応用することも一つの課題にしてもよいかもしれ ない。

HACK#49 の「オブジェクトを本当にポリモーフィックなものにしよう」に食欲がそそられた。モデルとビュークラスの作り方と使い方の簡単なサンプルが示されている。ここでは、Class:Trait というモジュールが紹介されている。「trait」というのは「特性」とか「形質」という意味である。

デスクトップCGIは、CGIをパイプ的あるいは連鎖的につないで実行する構想を当初から持っていたが、その発想はバッチ処理から抜け出していない。SQLデータベースによってAjaxなどを利用したインタラクティヴなシステムへの移行を進めているわけだが、次第にオブジェクト指向プログラミングとの親和性が高くなりつつあることを感じている。もっとも、@niftyの@homepageのCGIサービスはPerl5.005_02のままだからなあ。Perlを使って動的なページを試すなら、LaCoocanで遊ぶしかないかもしれない。

更新: 2007-10-21T11:19:24+09:00

12/22/2007 (Sat.)

[Perl] 言葉と物 C - たまねぎの状態 11

我がサイトにふさわしく、言葉と物シリーズ100番目の記事はPerlで飾ろう。Slashdot | State of the Onion 11ネタ。Programming is Hard, Let's Go Scripting...には、最近のLarry Wallの考えが示されている。

かなり詳細にスクリプティング言語について論じているので、興味のある人は読んでみるとよいだろう。結論として、スクリプティングの未来は、Perl 6にあるというのがLarry Wallの意見である。部分的に引用・訳出してみよう。

- まえがき

  私は、ほとんどの人々にとって、スクリプティングはわいせつなものに良く似ていると思う。私はそれを定義できないが、私はそれを見るとき、それを知るだろう。周囲に浮かんでいるいくつかの普通のミームがある。

  - 単純な言語
  - 「すべてのものは文字列である」
- 素早いプロトタイピング
- 糊言語
- プロセス制御
- コンパクト/簡明
- より悪いものはより良い
- ドメイン特定
  - 「電池内蔵」

  ...私はここの実際の中心を知らない。少なくとも技術用語で。私が一つの比喩を選ばないといけないとすれば、それは、「やさしい進入車線」(easy onramps)だろう。そして、遅い車線。たぶん、いくつかの選択的高速車線を持っている。

- Easy Onramps

  しかし、基本的に、スクリプティングは技術用語ではない。我々がなにかをスクリプティング言語と呼ぶとき、我々は初めに言語的文化的判断をしている。技術的

判断ではない。

私は人間性の一つのものとしてスクリプティングを見る。それは我々の言語的起源を示している。そう、ルーツについて話そう...

- The Past
- BASIC
- RSTS BASIC/PLUS
- Unix?
- JAM(no not that one)
- LISP
- Pascal, Ada
- Unix, shell
- BSD, csh
- C
- shell + awk + sed + find + expr...
- C xor shell
- Tcl
- Python
- Ruby
- *sh
- PHP
- JavaScript
- Monad/PowerShell
- Lua, AppleScript
- The Present

現在の状況を見ると、見えるものは様々なスクリプティングコミュニティであり、ジャングルの隣り合った部族かのように振る舞い、時には交易し、時には争うが、概してお互いの自己満足的孤立の方法には干渉しない。

私はこのようなことがらに人類学的な見地を取る傾向がある。あなた方の多くはPerlプログラマーであるか、いくらかの人は他のプログラミング部族から来ている。あなたの部族の歴史によれば、もしあなたがCプログラマーならば文字列をバイト配列のポインタと考え、関数プログラマーならばリストと、 Javaプログラマーならオブジェクトと考える。私は、文字列を大文字のTを持つテキストとして見る。

- Text

この項で言いたいことは少しわかりにくい。結局、言語学の素養のあるラリーらしい導入部なんだろう。Perlはポストモダン言語であるとか、すべての人間の言語はチューリング完全(Turing complete)であるとか、言語は言うことを強制するとかである。確かに言語は書き方や喋り方を書き手や話者に強制する。人間は修得した言語に従って世界を認知するからである。それがすべてではないが、プログラミング言語も同様である。以下の項が本論である。

- Wycliffe Bible Translators : World Missions for Unreached People Groups
- Turing completeness - Wikipedia, the free encyclopedia

それで、言語はあなたが言うことを強いられることにおいて異なる。明らかに、もしあなたの言語があなたに何か言うことを強いるとすれば、あなたはあなたの言語を使うあの特別な次元において簡明であることはできない。なにが、我々をスクリプティングに立ち戻らせるのか。

異なるスクリプティング言語が簡明であるためにいくつの方法があるのか?

ロシアのボルシチにいくつのレシピがあるのか?

言語デザイナーは多くの自由度を持つ。私はそれらのうちのいくつかを指摘したい。

- early binding/ late binding
- eager evaluation/lazy evaluation
- eager typology/lazy typology
- limited structures/rich structures
- symbolic/wordy
    - Huffman coding - Wikipedia, the free encyclopedia

- compile time/run time
- declarational/operational
- immutable classes/mutable classes
- class-based/prototype-based
    - Moose

本文の中で唯一のリンクである。

> Moose は Perl 5 のなかの Perl 6 なのですか?
>
> いいえ。Moose は Perl 6 に多くを触発されているが、Perl 6 それ自身ではない。代わりに、Perl 5 のためのオブジェクト指向システムである。私は、同じ旧い退屈な Perl 5 のオブジェクト指向コードを書くのに疲れ、Perl 6 のオブジェクト指向に涎が垂れたので、Moose を構築した。そう、Ruby に切り替える代わりに、私は Moose を書いた(^^)

(Moose - A complete modern object system for Perl 5, 著者: Stevan Little, 著作権: http://www.iinteractive.com,ライセンス: Perl と同じ)

クラスベースとプロトタイプベース両方のアプローチを可能にするために開かれるもう一つの次元がある。あなた方のだれかは、Self あるいは JavaScript のようなクラスのない言語に親しいかもしれない。クラスの代わりに、オブジェクトは原型からクローンをコピーするか、他のオブジェクトに委任する。多くの種類のモデリングでは、それは、実世界が働く方法に実際上近いものである。実際の有機体は再生産する時に DNA を複製する。自分自身 のいくつかの DNA を持たないと、@ISA 配列が、DNA の残りを含む親オブジェクトを告げる。

Perl 6のメタオブジェクトプロトコルのデフォルトはクラスベースであるが、プロトタイプベースのオブジェクトを仕立てるためにも十分に柔軟である。あなた方の うちの誰かはPerl 5のMooseを試したことがある。Mooseは本質的にPerl 6のオブジェクトモデルのプロトタイプである。とにかく、意味的なレベルにおいては。シンタックスは小さな違いである。望むべくは、Perl 6で少しでもより自然でありたい。

- Apocalypse 12: Objects
- Synopsis 12: Objects

- passive data,global consistency/active data,local consistency

データと制御についてのあなたの見方は、脳が如何に関数的にあるいはオブジェクト指向的であるかによって変化するだろう。人々は異なるように考え る。ある人々は証明可能な普遍的な真理によって、数学的に考える。関数プログラマーはすべてのものが純粋で副次効果から自由に見える限りは、スタックと ヒープを通じて明確な計算状態をばら撒くことはかまわない。

他の人々は、各自が個々の自由意志を持つ協同団体によって、社会的に考える。だから、どこかの連続したヒープにあることから離れて、各々の独立したオブジェクトに計算状態が格納されることはかなり重要である。

もちろん、我々のあるものは、我々がむしろ論理的なシャーロックホームズあるいは社会的なワトソン博士のいずれを模倣したいか、心に思い描くことが 出来ない。幸運にも、スクリプティングはこれらのアプローチのいずれとも非互換ではない。なぜなら両方のアプローチは通常の人々にとって、より取っ付きやすいからである。

- info hiding/scoping/attachment
- object/class/aspect/closure/module/template/trait
- transaction/reaction/dynamic scope
- process/thread/device/environment
- screen/window/panel/menu/icon
- syntactic scope/semantic scope/pragmatic scope
- use Lingua::Perligata;
- Curling Up

それで、私はすべてのスクリプティング言語はすべてのこれらの次元を考慮しなくてはならないということを示唆しているのではない。たとえ、Perl 6が試みているとしても。スクリプティングのパラダイムはこれらの次元のうちのいずれか一つではない。様々な理論によると宇宙は10か20の次元に横た わっているが、一般的に我々はそれらの次元の三つか半分ぐらいだけでなんとかやっていく。

我々がPerl 6と呼ぶスクリプティング言語の大部分は、時間の大部分を巻き上げた三次元の大部分を持つだろう。しかし、これらの次元を巻き上

> げないためには巨大な機械 を要する実宇宙に似ず、我々は宣言を正しく保つことによって次元を巻き上げないだろう。さて、我々は試みるだろう。そして、それが失敗するところでは、我々は、物事を正しく保つ文化を信頼するだろう。
>
> 例えば、それは正確にはPerl 5で既に起こったことである。我々は、宣言、use strict; use warning; を持っている。が、それらの使用を強制することを決めたのは文化である。そういう訳なので、我々は、Perl 6のほとんどではそれがデフォルトであるべきだと決めた。それはミツバチの群れによる決定のうちの一つである。この場合、群れ(swarm)は言語デザイナーよりも賢く方向転換した。そして、それはちょうど良い具合である。

Mooseモジュール(Moose-0.33)のSYNOPSISのサンプルには、「use Moose; # automatically turns on strict and warnings」とある。

- The Future

プログラミング言語の設計というものは言わば、事物をコンピュータ上で如何に表現し、事物の関係を如何に自在に取り扱うかに関わることになるのであり、その基盤となるものの設計なのである。プログラミング言語は仮想的に世界を記述する言語である。そして、その出力は現実空間さえ制御し、影響を及ぼすことになるだろう。

たまねぎの状態についてLWが真剣に語るのは初めてだろう。それだけ、苦しい時期なのだと思うし、いよいよこれならいけるという完成度に到達しつつある(Parrot released, perl6 passes sanity tests again, more!)ことも示している。是非、Parrot/Perl 6を完成して欲しい。Perl 5.10とMooseで遊んで待ってるよ。がんばれ。

- Perl6の今後 (2003/07/19)
- The State of the Onion 9 (2005/09/23)
- 日本Rubyカンファレンス2006 (2006/06/11)

更新: 2007-12-24T01:25:45+09:00

12/23/2007 (Sun.)

[プログラミング] オブジェクト指向プログラミング II - Meta Object Protocol

Mooseの次の出物はClass::MOPだ。Mooseはこのモジュール上に構築されている。

- オブジェクト指向プログラミング - 裏返しオブジェクト?!

12/29/2007 (Sat.)

[日記] 今年のまとめと来年の運勢

ある占いによると、2008年は僕にとって「決」の年で、自分をよく見直した上で決断する必要があるらしい。大体うっかりもので、ついつい馬鹿なことを口走るのが悪い癖なので、慎重にしないといけないと自戒することしきりであるが、三つ子の魂百までということなので、どうなるかわかったものではない。しかし、悪い年ではなく、発展的な良い年らしいので一安心。...「おいおい、とうとう運勢頼みかい」と影の声が・・・

今年はさてどんな年だったか、世間では、「偽」の年という残念な結果に終わったようだが、僕の場合は「本」だったかもしれない。新たに本を大量に買い、本棚から古い本を引っ張り出し続けた。本に埋もれて正月を過ごす「本正月」になるかもしれない。日記記事のうち、歴史を遡り、現在の源泉を探索し続け た「言葉と物」シリーズが100回を迎えた。その中で「モオツァルト」読解がまだ走っている。どんな具合に進展するのか、・・・一応の成果としては、SIMILE | Timeline を年表表示にも応用したこと(「日本のポストモダン」篇)。2か月分の日記記事の Timeline を生成するスクリプトも少し改良して、完全自動化達成^^)v

CodeZine - デスクトップCGI の 記事も、最後のまとめの段階に至って、現在のレベルでこれ以上進んでも技術的には繰り返しみたいな話になりそうなので、手が止まってしまった。SQL リ レーショナルデータベースと Ajax の導入によって、予定していた要素技術は基本的にはすべて出ているので、後は機能の設計やそれに対応したデザインとか インターフェースのような話になる。そのためにはフレームワークも必要だが、「言葉と物」シリーズの影響で、もう一度、スクリプト記法を含めて、スクリプ トの構成を見直したいという気持ちが強くなってしまった。

そうこうしているうちに、Larry Wall の The State of Onion 11 を読んだ(言葉と物 C - たまねぎの状態 11)ので、「Perl Hacks」や「Perl ベストプラクティス」で齧っていたオブジェクト指向プログラミングへの関心がさらに高まりつつある。Class-Trait-0.22: SEE ALSO にあるリンク、Traits — Composable Units of Behavior(形質 - 挙動の構成可能な単位)関連を調べている。Perl においては、Class::MOP(オブジェクト指向プログラミング II - Meta Object Protocol) や Moose などの著者である Stevan Little 氏の Class::Trait の Perl 5 実装があり、Perl 6 の仕様にある Roles が Traits の変種に当たるということだ。Class::Trait の後継が、Class::MOP や Moose になるのだと思われる。前述のリンクには多数の論文が示されている。実装があるのは、Squeak Smalltalk、Perl、Scala、C#と VisualWorks Smalltalk ということらしい。プログラミングも進化するのだなあと当たり前のことについて感動した。世界の捉えかたが時代とともに変化する。それによって、一般社会では言説が変化するのだが、コンピュータの世界においても同じようなことがあるということである。

言葉で記述された実世界のオブジェクトとプログラミング言語で記述されるコンピュータ上のオブジェクトの関係はどうなっているのだろうか。そんな取り止めのないことを考えながら今年も暮れていく。

> [更新: 実際は、物事はそれよりも少し自由である。role をクラスであるかのように使うならば、ちょうど role がするように匿名のクラスのインスタンスを自動的に生成する。クラスを role であるかのように使うならば、クラスの現在の状態の role 類似スナップショットを取り、匿名の role に凍結する。]

Larry Wall, Apocalypse 12, Classes, Date: 2004-04-13, Last Modified: 2006-05-25

Traits: to renew OO inheritance in a hacker style discussion(22 Feb 2004)には、「Traits — Composable Units of Behavior」の論文についての Larry や Stevan Little らの議論がある。Apocalypse 12 はそれを踏まえているからか、traits の用語もある。role のアイデアはそれ以前からあった。

更新: 2007-12-29T15:06:20+09:00

2/10/2008 (Sun.)

[オブジェクト指向] 言葉と物 CVII - モデリングと抽象化

先週の木曜日は、朝4時半起きで、午前10時には東京でメールで届いた資料を修正して、ミーティング。帰りには、種村季弘著「書物漫遊記」(ちくま 文庫、1986;初出: 筑摩書房、1979)と吉本隆明

著「柳田国男論・丸山真男論」(ちくま学芸文庫、2001)を手にしていた。最近は家内から読む本が支離滅裂だと言われる。なにしろここ数年前までは、コンピュータ・プログラミング関係か、ブルーバックス、SF、クランシー、立花隆、量子論、宇宙論、複雑性の科学、脳科学、人工知能関連ぐらいしか買わなかった。

この読書傾向の変化は、この日記がもたらしたものだ。新しい知識が世界を覆いつくすまでは終わらないだろう。

[本]『 複雑な世界、単純な法則 』 - Small World 再び経由、SS > NF reviews > Mark Buchanan 経由、 Susan Stepney: home にある modelling and abstraction が今日のタイトルだ。このページにある An OO structuring for Z based on Views が具体的なものだ。ここの Views は、Model-view-controller(日本語)の view だ。ここにある Structuring(構造化)は単純なオブジェクト指向とは違う感じ。

「Z」とは何かを探るのが本日の任務なのだが、Wikipedia にもページがある。

- Z notation - Wikipedia, the free encyclopedia
- Z 言語
- Zermelo-Fränkel set theory
- axiomatic set theory
- 公理的集合論
- 定義 - Wikipedia

結局、世界を覆いつくすには集合論を理解しなくてはならない。

Z は仕様記述言語の一種で、UML の形式的なチェックに使おうとの試みもあるようだ。言葉と物の立場からは、ものごとの構造のモデルを考えるのには参考になるかもしれない。

更新: 2008-02-17T17:46:01+09:00

2/23/2008 (Sat.)

[オブジェクト指向] 言葉と物 CIX - Moose の研究 I まえがきなど

Larry Wall も注目している最新のオブジェクト指向システムモジュール、Moose を使ってみる。題材として、手書きの Walking World Project Google Maps API v2 版を CGI で書き直すことにした。これを整理して、5 月創刊予定の TSNET 会報に載せる予定だ(投稿規程)。ここでは下書きのようなものになるだろう。

僕が Perl でオブジェクト指向を試みたのは、「実践実用 Perl」(2004 年)に「オブジェクト指向 Perl モジュールミニ入門」を書いた時が 最初である。自分のために書いたのである。その頃は、リファレンス-デリファレンスさえ、よく知らなかったものだ。まだ、Perl4 のプログラミング・スタイルから抜け出していなかったから当然だった。単なるテキスト処理にオブジェクトは必要とは思われなかったが、Web クライアントプログラミングで Perl5 のオブジェクト指向モジュールを使い始めていた僕はその必要性に気付いていた。「ミニ入門」を書くことによって、ようやく Perl5 に近づいた のだった。

昔ながらの bless を使う工作的なオブジェクト指向モジュールに比べ、Moose を使ってオブジェクト指向モジュールを書くのは超簡単でスマート だ。new メソッドを定義する必要もない。オブジェクトの属性を列挙して、必要な属性を使って、メソッドを書くだけ。例えば、Walking World Project ページのヘッダ、ボディとフッタ出力用のモジュールはこんな感じ。オブジェクト属性値の読み書き可否、データ型、デフォルト値などを指定で きる。

```
package CreateGMap;
use Moose;

has 'baseurl' => (is => 'rw', isa => 'Str', default => '');# マーカーデータ中の相対URL補完
用
has 'key' => (is => 'rw', isa => 'Str', default => '');# Google Maps KeyのURL文字列
has 'title' => (is => 'rw', isa => 'Str', default => '');# ページタイトル
has 'desc' => (is => 'rw', isa => 'Str', default => '');# ページの説明
has 'id' => (is => 'rw', isa => 'Str', default => 'map');# Google Mapの表示位置
has 'width' => (is => 'rw', isa => 'Int', default => 500);# Google Mapの幅
has 'height' => (is => 'rw', isa => 'Int', default => 600);# Google Mapの高さ
has 'messageid' => (is => 'rw', isa => 'Str', default => 'message');# Google Mapのセンター
経度緯度表示
has 'lng' => (is => 'rw', isa => 'Num', default => 132.472833);# 経度
has 'lat' => (is => 'rw', isa => 'Num', default => 34.385555);# 緯度
has 'data' => (is => 'rw', isa => 'Str', default => 'data.xml');# マーカーデータファイル
has 'zoom' => (is => 'rw', isa => 'Int', default => 1);# Google Mapの縮尺設定

sub header {
        my $self = shift;
        my $header = <<HEADER;
.....
        <title>$self->{title}</title>
.....
HEADER
        return $header;
}

sub body {
        my $self = shift;
.....
        return $body;
}

sub footer {
        my $footer = <<FOOTER;
.....
        return $footer;
}

1;
```

次に、Google Mapsで表示する地域を切り替えるデータリストの出力用モジュール、CreateGMap::Area。

```
package CreateGMap::Area;
use Moose;
```

```
has 'type' => (is => 'rw', isa => 'Str');# button
has 'name' => (is => 'rw', isa => 'Str');# button 名
has 'value' => (is => 'rw', isa => 'Str');# ボタンに表示する地域名
has 'lat' => (is => 'rw', isa => 'Num');# 緯度
has 'lng' => (is => 'rw', isa => 'Num');# 経度
has 'data' => (is => 'rw', isa => 'Str');# Marker 用データファイル名
has 'zoom' => (is => 'rw', isa => 'Int');# Google Map の zoom 値
has 'desc' => (is => 'rw', isa => 'Str');# 地域紹介

sub tableunit {
        my $self = shift;
        return sprintf("¥t<tr><td>¥n¥t<input type=¥"%s¥"
 name=¥"%s¥" value=¥"%s¥" onClick=¥"onLoad(%.6f, %.6f, '%s',
 %d);¥">¥n<p>%s</p>¥n<hr>¥n¥t</td></tr>¥n",
 $self->type, $self->name, $self->value, $self->lat, $self->data,
 $self->lng, $self->zoom, $self->desc);
}

1;
```

こんな具合に使う。

```
#!/Perl5.8/bin/perl.exe
use CreateGMap;
use CreateGMap::Area;
use YAML qw(LoadFile);

my $gmap = CreateGMap->new(
        baseurl => '<- リンク URL が相対的な場合に他のサイトに表示する場合に使う ->',
        key => '<- Your Google Maps Key ->',
        title => '<- Page Title ->',
        desc => '<- Page description ->'
);
print $gmap->header;
print $gmap->body;
my @hashref = LoadFile('<- YAML 形式の地域切替用データファイル ->');
foreach (@hashref){
        my $area = CreateGMap::Area->new(%$_);
        print $area->tableunit;
}
print $gmap->footer;
```

確かにモジュールを使って書くとスクリプト本体は簡潔に書けるようになる。現実の複雑さがオブジェクト指向モジュールによって隠される。オブジェク ト指向モジュールを作成する過程で、問題解決に必要な情報と結果となる表現とが整理されて結び付けられる。しかし、モジュールの使い方を覚え、スクリプト を書いて目的を達するだけではプログラミングの技量はほとんど向上しないだろう。是非、自作のモジュールを生み出すべきなのである。それには、Moose を使うのが最適かもしれない。

- Stevan Little / Moose - search.cpan.org
- Moose - A postmodern object system for Perl 5 - search.cpan.org
- オブジェクト指向 - TSNETWiki

更新: 2008-02-24T11:02:31+09:00

2/24/2008 (Sun.)

[オブジェクト指向] 言葉と物 CX - Moose の研究 II ポストモダン

Perl は何の略かというと、二つぐらいが有名だが、Postmodern programming language の略だと言えるかもしれない。s/(P)ostmod(er)n programming (l)anguage/$1$2$3/だからであるというのは冗談だが、ポストモダンのプログラミング言語だとは、LW が言っているはずだ。Moose を、作者の STEVAN Little 氏はポストモダンなオブジェクト指向システムだと言っている。Perl は 1987 年に生まれた。1989 年にベルリンの壁が崩壊したのを近代 の終わりと見做そうという考え方もあるらしいので、ポストモダンなプログラミング言語と Larry Wall 氏が何時発言したのかは確認していないが、適切な考え方かもしれぬ。ポストモダンなオブジェクト指向システムが出るのにはさらに 20 年程度を要し たことになる。

Moose とは、postmodern object system の略である。s/post(mo)dern (o)bject (s)yst(e)m/$1$2$3$4/だからである。意味は「へらじか」。現在、頻繁にバージョンが上がって、モジュールが増殖中である。まだまだ、 ドキュメントに一貫性がないが、先に見たように簡単にオブジェクト指向モジュールを作れることは間違いない。Walking World Project の Google Maps v2 版は、Moose を使ったスクリプトが出力したものに置き換えた。なにやら期待を持たせてくれる。古いモジュールを Moose で拡張することさえできるみたいなことが書いてある。しかし、それにはかなり深いレベルでの理解が必要なようで、もう少し詳しいサンプルが欲しいところだ。

3/1/2008 (Sat.)

[オブジェクト指向] 言葉と物 CXI - Moose の研究 III モダンの未来

TSNET で、Larry Wall がどこで最初に Perl は「ポストモダンのプログラミング言語だ」と言ったかが話題となった。

オライリーの perl.com にある Perl, the first postmodern computer language の 記事(Larry Wall,March 09, 1999)には、「The following is the text of Larry Wall's talk at Linux World on Wednesday, March 3.」と書かれていて、それが正解である。実は、Moose モジュールのドキュメントにもそのことが記載されているので翻訳してみよう。

> **待って、*ポストモダン*、私はちょうど*モダン*だと思ったのだが?**
>
> つまり、私は Larry Wall の「Perl, the first postmodern computer language」と題した 1999 年の Linux World のトークを読んだのだ。そこで、彼は、彼が如何にクールと考えるが故に Perl のための機能(features)を取り上げ、価値がないと考えるものを捨てたかについて話した。これは、我々が Moose で如何に同じことをしたかについて考えさせた。Moose のために、Perl 6、CLOS(LISP)、Smalltalk、Java、BETA、Ocaml、Ruby などから機能を「拝借」した。そして、我々が好きでない(価値がない理由)部分をポイと投げ捨てた。

つまり、この理由によって(そして二、三の他の理由で)、私は再び、Moose をポストモダンオブジェクトシステムと呼んだ。

もう十分。

(Wait, post modern, I thought it was just modern?)

Larry Wall は Postmodern だけでなく、哲学や歴史にもかなり詳しい。少し勉強して読まないと歯が立たないかも。Larry Wall のトークは 1999 年で、1997 年のソーカル事件(「知」の欺瞞)を受けた話だろうと思われる。しかし、僕らは LW を含めて 1970 年代に Postmodern の元となる構造主義を知っていた年代なのだ。もっとも当時今理解している程度のレベルにあったとは言い難い。ファッションに過ぎな かったかもしれない。Larry は the Modern period は間違えた名前付けだと言う。モダンの概念は現在(now)や新しい(new)概念を常に連想するからだ。そうするとポストモダンは未来を指 すべきだというのが本記事のタイトルである。日本語では、「近代」と「現代」という用語を使い分けているが、英語には modern しかない。しかし、歴史 が進行するにつれて、近代も近代でなくなり、現代も現代でなくなるから、結局同じ矛盾を孕むようになるわけだが・・・そのうち、人間は古典主義時代以降の 近代に新しい名前を考え出すだろう。現代との区分をどこにするかはまだ定かでない。

- ソーカル事件 - Wikipedia
- 『「 知」の欺瞞 』関連情報
- 脱構築
- マルティン・ハイデッガー
- ジャック・デリダ

ポストモダンについては今後取り上げる機会もあるので、とりあえず放り投げておいて^^;)、BUILDING CLASSES WITH MOOSE の項を訳しておこう。

Moose でクラスを構築する

Moose はクラス構築/定義に、できるだけ多くの便利さをもたらすようにあらゆる試みを行うが、あなたが欲するなら、あなたの方法の外に止まる。ここに Moose でクラスを構築する時、注目すべき二、三の項目がある。

extends で特定されないならば、Moose を使うあらゆるクラスは、Moose::Object から継承する。

Moose は継承する属性を含めすべての属性を has で定義して管理する。そして、Moose::Object から継承する new を呼ぶことを仮定し、すべてのインスタンス・スロットを適切に初期化し、適当なデフォルト値をセットし、型制約をチェックして、強制する。

(BUILDING CLASSES WITH MOOSE)

- 10/18/2007: [サイバースペースカウボーイズ] オブジェクト指向
- 10/20/2007: [プログラミング] オブジェクト指向プログラミング - 裏返しオブジェクト?!
- 12/22/2007: [Perl] 言葉と物 C - たまねぎの状態 11
- 12/23/2007: [プログラミング] オブジェクト指向プログラミング II - Meta Object Protocol
- 12/29/2007: [日記] 今年のまとめと来年の運勢 (2007/12/29)
- 02/10/2008: [オブジェクト指向] 言葉と物 CVII - モデリングと抽象化

- 02/23/2008:［オブジェクト指向］言葉と物 CIX - Mooseの研究 I まえがきなど
- 02/24/2008:［オブジェクト指向］言葉と物 CX - Mooseの研究 II ポストモダン

更新: 2008-03-07T21:55:59+09:00

3/2/2008 (Sun.)

［オブジェクト指向］言葉と物 CXII - Mooseの研究 IV テキスト

なぜ、最近、なぜLarryがPerlがポストモダン言語と言ったはずと思ったのかを思い起こすと、言葉と物 C - たまねぎの状態 11に書いたからだ。もう忘れているˆˆ;)

「プログラミングは難しい、スクリプティングをしよう...」(Larry Wall、Programming is Hard, Let's Go Scripting...、Oreilly perl.com THE SOURCE FOR PERL、2007-12-06)の「Text」の項を訳出しておこう。

最初が大文字の「テキスト」

私はポストモダンの観点から言語を読む。もちろん、ポストモダンの用語はそれ自身、文脈に敏感である。ある人々はポストモダニズムは民衆の権利拡大と大差ないと考える。ある人々はPerlについて同じことを意味すると考える。

しかし、私は、ポストモダニズムは、話されるか、書かれたいずれかのTextが両端の知性を、そして時にはその中間の知性を必要とするコミュニケー ション活動であることを意味すると取る。私はおろかなコンピュータ言語に話しかけたくない。私は(私が)タイプした文字列を理解する(私の)コンピュータ 言語が欲しい。

Perlはポストモダン言語である。そして、多くの保守的な人々はポストモダニズムはむしろ進歩的な概念のように感じる。だから、ポストモダニズム に関する私の物の見方が、宣教師、特にthe Wycliffe Bible Translatorsに教えられるように言語学や翻訳を勉強することによって知識を与えられたことは皮肉である。彼等が力説したことの一つは、実際、未 開の人間言語のようなものはないということである。彼等が本質的に主張するところは、すべての人間言語は「チューリング完全」であるということである。

あなたが、いわゆる未開部族のところに行き、彼等の言語を分析すると、構造的にそれが他の人間言語と同程度に複雑であることを見出すことになる。基本的に、あなたが十分に長く使用すれば、人間言語でかなり多くの事柄を話すことができる。強いて言うなら、人間言語は「チューリング完全」である。

人間言語はそれゆえにあなたが言うことができることにあるのではなくむしろ、あなたが言わなければならないことにある。英語では、単数と複数を区別 することを強いられる。日本語では、単数と複数を区別しなくてよいが、話しかける人への尊敬の度合いだけでなく、話す人や物についての尊敬の度合いを考慮して、特定のレベルの礼儀正しさを取らなければならない。

(以下は既に訳した。→ 言葉と物 C - たまねぎの状態 11)

放り投げたはずのポストモダンが思いがけず戻ってきてしまったので、今度は地球周回軌道まで打ち上げておいて、オブジェクトの話に戻ろう。ドッカーン・・・3万6千キロメートル上空へ。ポスト

モダンはしばらくは戻ってこないだろう。

最新のたまねぎトークの「immutable classes / mutable classes」の項を訳してみよう。

不変クラス / 可変クラス

Java のクラスは閉じられている。その理由の一つは Java がかなり速く走ることができることである。逆に、Ruby のクラスは開かれている。それが意味するところは、あなたはいつでも新しいものをそれに付け加えることができることである。開く選択を維持することの理由の一つはたぶん Ruby が ゆっくり走ることである。が、その柔軟性は Ruby が Rails を持つ理由でもある。

Perl 6 は不変総称(generics)と可変クラスの興味深い混合をここに持ち、時にクラスを閉じることを可能にすることについて興味深いポリシーを持つだろう。例えば、クラスはそれ自身を閉じたり、終了させることはけっして許されない。ごめん、ある理由で私は Perl 6 について話し続けている。それは Perl 6 をデザインするすべてのこれらの次元について考えなければならなかったという事実と関係がある。

(Programming is Hard, Let's Go Scripting...)

この項は、Moose が登場する「class-based / prototype-based」の項の前にある。

- 総称型 - Wikipedia

総称とは、データ型を無指定にしておけるクラスで、データ型は継承したくないようなクラスに用いるらしい。Perl 5 ではデフォルトみたいなものだから、OOP の第 12 章総称、12.1 のタイトルは「Perl に特別な総称メカニズムが不要な理由」となっている。型のあ るPerl 6 ではそういうことでもないのだろうけど、Synopsis 12: Objects には generics という項目は見当たらないが、・・・role がその役割を担うのかな?・・・Open vs Closed Classes という項目はあるが・・・

- Inferring (Foo of Int).does(Foo of Any)

更新: 2008-03-02T23:09:10+09:00

3/7/2008 (Fri.)

[オブジェクト指向] 言葉と物 CXIII - Moose の研究 V プロトタイプ

Perl ではサブルーチンを呼び出すときに、&をサブルーチン名の先頭に付ける。それが、なぜか、&を付けなくても呼び出せるよう になった。このようなしゃべり方もやはり Perl 4 プログラマから抜けきっていなかった僕の状態を表す現象の一つである。もっとも気付いたのはだいぶ以前の話ではあるのだが。モジュールのメソッドを呼び出す場合(関数呼び出し)がその典型である。しかし、なぜ、そんなことができるのかは理解していなかったのである。これは旧式のサブルーチンの話ではなかったのだ。

サブルーチンはプロトタイプを利用すれば、サブルーチンを宣言する場合に、引数の個数や型に制約を加えることができる(プログラミング Perl 改訂 版、2.7 サブルーチン、2.7.3 プロトタイプ、133 ページ;プログラミング Perl 第 3 版 Vol.1、6 章、6.4 プロトタイプ、261 ページ)。Larry が「Moose は本質的に Perl 6 のオブジェクトモデルのプロトタイプである」としたのは、このプロトタイプのことを指しているらしい。プロトタイプのサブルーチンとして呼び出す場合に は、&を付けずに呼ぶ(関数呼び出し)。&を付けると制約がなくなる。これは総称メカニズムと関係があるのかな・・・

OOP (Object Oriented Perl; ダミアン・コンウェイ著「オブジェクト指向 Perl マスターコース オブジェクト指向の概念と Perl による実装方法」、ピアソン・エデュケーション、2001 年)ではプロトタイプのメカニズムの話は出ていないようだ。ちなみに Perl にプロトタイプが実装されたのは、5.002 からである(perlhist - the Perl history records には「Larry 5.002 1996-Feb-29 Prototypes.」とある)。

ここで、Programming is Hard, Let's Go Scripting... の「early binding / late binding」を少し訳してみよう。

早い束縛 / 遅い束縛

> この文脈の束縛は、あなたが任意のルーチン名で呼び出そうとするルーチンを決定する時点に生ずるのが大体正確なところである。コンピューティングの初期においては、ほとんどの束縛は効率の理由からかなり初期になされた。コンパイル時か、遅くともリンク時である。静的に型付けされた言語にまだこのアプローチを見ることができる。しかしながら、我々は Smalltalk のような言語に異なった傾向を見始めた。そして、今日では、ほとんどのスクリプティン グ言語は遅い束縛に向かう傾向がある。それが、スクリプティング言語が dwimmy(どういう意味^^;)である(意味するところをする)ことを試みる理由であり、最も意味のあることをする(dwimmiest な)決定は、通常は遅い決定になる。なぜなら、より利用できる意味的、実用的文脈を持って作用するのは、その時だからである。さもなければ、あなたは未来を予言しなくてはならない。それは難しい。
>
> それで、スクリプティング言語は自然にオブジェクト指向の視点へ動いていく傾向がある。そこでは、メソッドが仕事を片付ける時間まで束縛は生じな い。C++や Java のような言語に衝突の傷をまだ見ることができるが、C++はデフォルトのメソッド型を非仮想的に作るので、あなたは仮想的に明確に、 遅い束縛を得ることを言わなくてはならない。Java は最終クラスの概念を持ち、本質的にコンパイル時に呼び出しをクラスに結びつけることを強いる。私は、両方のこれらのアプローチは大きな過ちだと考える。Perl 6 は異なる過ちを犯すだろう。Perl 6 においては、すべてのメソッドはデフォルトで仮想的である。全体として、アプリケーションのみが、クラスを仕上げるようにオプティマイザに告げることが できるが、おそらく、あなたが、いかにすべてのクラスが、プログラムにおいてすべての他のモジュールによって使われようとしているかを知った後である。

この部分に Smalltalk が出てくるが、「Prototype-based Object」で検索すると、Vladimír Janoušek 氏の「On the Prototyped-Based Object Orientation in Systems Modeling and Simulation」が引っ掛かる。ここでは、C++や Java で実装された DEVS というシステムを Smalltalk(実際には Squeak)で実装して、より柔軟なアプローチをサポートできることを示している。Smalltalk の拡張、package Prototypes を使う。これが Smalltalk について Larry の言っていることだろう。

- Ing. Vladimír Janoušek, Ph.D.
  - Research
    - Papers

更新: 2008-03-09T09:19:13+09:00

[オブジェクト指向] 言葉と物 CXIV - Moose の研究 VI 複雑な構造

物理学は数学言語を使って、素粒子から宇宙までのスケールの時空の構造を描き出す。例えば、言葉と物 LXXXVII - E8 の 記事にあるように。超ひも理論は 10 次元の理論であるが、次第に理論は複数

あることがわかってきて、それを統合するのが、11 次元の M 理論である。M は母 (Mother)の M である。ローレンス・M・クラウス著『超ひも理論を疑う - 「見えない次元」はどこまで物理学か?』(早川書房、2008 年)では、超ひも理論は次元の巻上げを疑われているようだが、Perl 6 は宣言をまっすぐに保つことによって次元を巻き上げずにおけるらしい。プログラミング言語は何を描き出すのだろう。

テキスト処理の世界はインターネットがもたらされて以来、大きく変化した。それ以前のテキストは基本的にはプレーンテキストだった。初期のテキスト 処理は特殊なマークアップによってワードプロセッシングを実現するような仕組みが中心だった。テキストを XTR+TBL や FIN のようなツールで処理する と、Excel などで作られたような複雑な表を含む整形された文書がページプリンタから出力された。そのこと自体が驚きだった。1980 年代後半のことだったろう。1990 年代に入って、テキスト処理の素材の中心となっていったのは、パソコン通信のログだった。1990 年代、NifTerm や AirCraft などの NIFTY-SERVE のブラウザが全盛を迎えるとほぼ同時に、WWW が一般ユーザーに普及しはじめる。Windows の時代に なって、僕にとってはスクリプティング言語はデスクトップのツールとして行き場を失いつつあった。DOS 窓ではエスケープシーケンスが機能しなくなったからだ。

CGI が登場したのはそれほど昔でもない。インターネットが普及したのと同時期、1995 年ぐらいだったらしい。最初に CGI を知った時はそれほど 興味は持てなかった。まだ HTTP だの、LWP モジュールなど知らなかったせいもある。パソコン通信のログのほうが優勢な時代だったのである。HTML でマークアップされたテキストが、Web に溢れるようになったのは、21 世紀に入ってからだろう。LWP::Simple モジュールの get 関数で、任意の URL の HTML テキストをスクリプトで読み込めると知った時は、インターネット時代のテキスト処理の大きな可能性を感じたものだ。それがデスクトップ CGI となって「実践実用 Perl」を生み出した。スクリプティング言語の主戦場は、デスクトップにおいても Web ブラウザとなったのである。少なくとも 僕にとっては・・・

テキスト処理は、行単位でテキストを読みながら、パターンマッチで必要な情報を抽出し、一塊のデータとしてまとめながら出力する技術である。他の情 報を付加した合成出力も可能だし、プログラム自身を出力することさえ可能である。テキストからなるものを生み出すことにおいては、事実上不可能なことはない。おそらく、データの入力や実行制御の方法に関わるインターフェースとデータをどのように保持するかに関わるデータベース化が、フレームワークとしてシ ステム化するためには重要な部分だろう。

話は複雑な構造に行き着くはずだった。Perl 4 では、一塊のデータを、単にタブで区切られた文字列の値と一意となる文字列をキーとする連想配列とし、DBM に格納するのが常套手段であった。それが MySQL のような RDB が自由に使えるようになって、少し使い始めたが、データベースの生成のところが面倒くさい。それで、MLDBM にデータを複雑な構造のまま格納するという話を書く予定だった。しかし、複雑な構造とは何だろうというところで、気分は他のところに飛んでいってしまったのだ。

- perl .com: Managing Rich Data Structures

この複雑な構造とやらで何をするんだい、うーむ、必然性を感じられるネタがないかなとしつこく考えている^^;)

更新: 2008-03-16T10:03:24+09:00

3/9/2008 (Sun.)

[オブジェクト指向] 言葉と物 CXV - Moose の研究 VII メソッドのプロトタイプ

『ダミアン・コンウェイ著「オブジェクト指向 Perl マスターコース オブジェクト指向の概念と Perl による実装方法」(ピアソン・エデュケーション、2001 年、以下 OOP)ではプロトタイプのメカニズムの話は出ていな いようだ。』では「総称」の章を探しただけだったので、インデックスで「プロトタイプ」を調べた。

オブジェクト指向のメソッドでは、サブルーチンのプロトタイプは無視されるというのが結論である(3.2 Perl の単純なクラス、「メソッドのプロトタイプ」、119 ページ)。オブジェクト指向モジュールでは関数呼び出しをしないことで、プロトタイプは有効にならないことになる。なあるほど、そうなのか。「Perl コンパイラがプロトタイプをチェックできないのは、Perl のすべてのメソッドが多態的である ためだ。・・・」としていて、第 6 章「継承」にその説明がある(6.1.2 Perl での継承の意味、234 ページ)。

それはともかく、やはり Damian Conway 著である「Perl ベストプラクティス」(オライリー・ジャパン、2006 年、以下 PBP)では、副作用のあるサブルーチンのプロトタイプは 使うなということになっている。「オブジェクト指向実装では、どのメソッド呼び出しであっても完全に無視されるために、無害であるかのような錯覚を抱かせる」とある(9 章 サブルーチン、9.10 プロトタイプ、212 ページ)。

結局、Perl のオブジェクトはどのように使うのか。OOP(2001)では、クラス生成の自動化(第 8 章)に Class::Struct と Class::MethodMaker が紹介されているがまだ物足りない感じ、PBP(2006)では、Class::Std が一貫して紹介されている。 Class::Std は inside-out オブジェクトを取り扱えるのだろう(オブジェクト指向プログラミング - 裏返しオブジェクト?!)。そして、Perl 6 の影響を受けたポストモダンオブジェクトシステム、Moose/Class::MOP の登場となったということなのかな。

キーワードはメタクラスで、Class::MOP は Smalltalk や CLOS(Common Lisp Object System)の影響を受けている。ざっくりとした Perl のオブジェクト指向の流れは掴めたので、そろそろ Moose/Class::MOP へ戻ろう。

更新: 2008-03-13T22:46:54+09:00

3/16/2008 (Sun.)

[オブジェクト指向] 言葉と物 CXVI - Moose の研究 VIII メタクラス

Moose や Class::MOP のドキュメントには様々な論文が参照文献として記載されている。そういう文献を調べてみると、主に Squeak などの Smalltalk 系言語が使われている。

オブジェクト指向プログラミングを勉強する場合に最初に閉口するのは、知らない用語が多すぎることである。しかし、これには慣れていく必要がある。 クラスとは何か、オブジェクトとは何か、そしてメタクラスとは何か。その他にも、様々な用語が溢れ出し、溺れそうになる。そのような用語を自在に使えるようになる時は、オブジェクト指向を自家薬籠中のものとできたときだろう。
Class::MOP のドキュメントが参照している「Uniform and safe metaclass composition」(Stéphane Ducasse, et al., "Computer Languages, Systems & Structures", 31, 2005, 143-164)の要約から少し引用訳出してみよう。

> 純粋なオブジェクト指向言語においては、クラスはオブジェクトであり、メタクラスと呼ばれる他のクラスのインスタンスである。クラスがそのインスタンスのプロパティを定義するのと同じ方法で、メタクラスはクラスのプロパティを定義する。それゆえに、クラスのプロパティを再使用しようとすることは非常 に自然であり、いくつかのクラスの間でそれを利用することができる。しかしながら、これは、「メタクラス構成問題」を引き起こす。すなわち、あるクラスに 適用されたコード断片が、それぞれのメタクラスの継承関係により、他のクラスに適用された時に壊れてしまう。

Moose/Class::MOP の作者の Stevan Little 氏は、その前に Class::Trait を書いた。Traits という概念は重要だと思われる。Perl 6 の Synopsis 12: Object では、trait という用語は普通名詞で使われる。trait は遺伝学では形質という意味だ。クラスのプロパティに近い意味で使われる。より基本的な単位、挙動の構成可能な意味的な単位となるべきものなのだろう。

Class::Trait の参照文献「Traits: The Formal Model」のやはり要約の最初の部分を引用訳出しよう。

> Traits は挙動の再利用単位であり、オブジェクト指向プログラムにメソッドのレベル以上ではあるが、クラスのレベル以下の構造化のレベルをもたらす。

そろそろ継承の問題に足を踏み込まなくてはならない。Class::MOP の MOP は「Meta Object Protocol」の略である。複数のオブジェクトには約束事を持ち込まねば、メタクラス構成問題が生じるのは当然のことである。いよいよオブジェクトも人間臭くなってきた。Stevan Little 氏は Moose を最初、Complete Modern Object System と呼んでいたのだが、結局、Postmodern と呼ぶことにした。人間は完全ではありえないが、完全を求め続けるのである。未来に向けて。

更新: 2008-03-16T22:54:14+09:00

3/18/2008 (Tue.)

[オブジェクト指向] 言葉と物 CXVII - Moose の研究 IX Ruby

何か日曜日にだけ書くみたいな日々が続いている。やれやれ。お疲れだが。うむ、オブジェクト指向の研究に Perl だけを見るのもなんだなと思い、「Python 入門」(1998)、「Python プログラミング」(1998)、「初めての Python」(2000)を見なおしてみる。しかし、これらの入門書ではそれほどオブジェクト指向を売り物にしていない。class は便利だから使いなという程度。確かに Python がオブジェクト指向だという意識はそれほどない。やはりオブジェクト指向は Ruby かなと、その名も「オブジェクト指向スクリプト言語 Ruby」(1999)を引っ張り出す。

オブジェクト指向について勉強するには大変詳しく実際的なので興味深い。プロトタイプベースのオブジェクト指向や「クラス合成」の問題についても言及がある。書かれたのが 1999 年なので、Traits についてはない。ruby-talk を「traits」で検索した結果のトップ 2 件をピックアップ。調べてみると、traits-0.0.0 が出たのが 2005 年である。Ruby-Traits は 2007 年リリース。

- [ANN] traits -0.10.0, Author: ara.t.howard noaa.gov, Date: Fri, 6 Oct 2006 04:23:08 +0900
- [ANN] Ruby- Traits 0.2 released (sorry long), Author: Robert Dober, Date: Wed, 24 Oct 2007 05:16:48 +0900

Wikipedia で Trait (abstract type)を 見ると、Ruby の Module mixins はある程度 traits に似ているとあった。「オブジェクト指向スクリプト言語 Ruby」の Mix-in(177 ページ)には、Mix-in クラスは、「インスタンスを持たない抽象クラス」であり、かつ「Mix-in 以外のスーパークラスを持たない」とある。

まつもとさんの 2004 年の日記に Traits への言及があった。

- [OOP]Applying Traits to the Smalltalk Collection Classes
- [言語]Traits

更新: 2008-04-03T00:18:00+09:00

3/19/2008 (Wed.)

[オブジェクト指向] 言葉と物 CXVIII - Moose の研究 X プロトタイプベース・プログラミング

オブジェクト指向は奥が深い。Prototype-based programming を調べた。クラスのないオブジェクト指向プログラミングの一種ということらしいのだが、・・・。Wikipedia の Languages の項のリストの Perl の部分を見ると、Class::Prototyped のモジュールへのリンクがある。このモジュールの see also には、Class::Classless へのリンクもある。

- Prototype-based programming - Wikipedia, the free encyclopedia

Synopsis 12: Objects には『Perl 6 においては、クラスオブジェクトを含め、すべてのオブジェクトはそのメタクラスのインスタンスに委譲する。HOW メソッドによってオブジェクトのメタクラスを得ることができる。「クラス」のオブジェクトは「空(から)」のインスタンスと考えられ、より適当には「プロトタイプ」オブジェクト、あるいは「プ ロトオブジェクト」と呼ばれる。』とある。Perl 6 では、Traits やメタクラス・プログラミングの概念がプロトタイプに結びついているように見える。これが、Moose が Perl 6 のプロトタイプと言われる所以である。あまり用語に拘っているとわけがわからなくなるかもしれない。

Perl 6 Design Minutes for 05 March 2008 の 話によれば、Larry が、Perl 6 の標準文法を Perl 5 で動かそうとしている!? Perl 5 の型システムの制約がオブジェクトを獲得するのを難しくしていて、Moose を使うというアイデアについての Larry のコメントもある。Larry は perl5 to perl6 translator を書いているはずだが、逆の方向の話が出ているのはなぜだろう。Perl 6 Design Minutes の断片的な会話から意味を取るのは大変難しい。おそらく、そのような translator は a2p のようなわけにはいかない。Perl 5 のなかで、Perl 6 を表現できるようにすることが translator には最低必要なのかもしれない。そして、その時には、C 版の Perl 6 ができるということであるかもしれない。

- PPI and the Perl 5 to Perl 6 converter ?
- is perl5 -Dmad output used in perl 6 ??
- A proposition for streamlining Perl 6 development

更新: 2008-03-20T23:52:39+09:00

3/20/2008 (Thu.)

[オブジェクト指向] 言葉と物 CXIX - Moose の研究 XI オブジェクトを巡って

昨年末の Larry の「プログラミングは難しい、スクリプティングへいこう」の記事で、Moose に出会って、オブジェクト指向を見直すきっかけと なった。プロトタイプベース・プログラミングは、クラスのないオブジェクト指向プログラミングというよりは、メタクラスのあるオブジェクト指向プログラミ ングとなりつつある。メタクラスはクラスのプロトタイプを生み出す。Perl のサブルーチンのプロトタイプ(関数指向)がオブジェクト指向によって一般 化・拡張されたものであると見做せるのかもしれない。

Perl 6 の HOW メソッドによって、Larry が予感しているように、メタクラスは考えうる限りだけ存在するだろう。世界を被い尽くすまで。Stevan Little 氏が Moose を完全にモダンなオブジェクトシステムと最初は豪語していたのを後で謙虚にポストモダンと言い換えたように、世界の見方はいくつもある、超ひも理論がいくつもあるように。Moose/Class::MOP の発展が今後の大きな楽しみである。

さて、ここから、ミシェル・フーコーへと展開する道もあるが、それはまた別の楽しみとして取っておいて、Moose 実践実用の思索に戻ろう。

3/21/2008 (Fri.)

[オブジェクト指向] 言葉と物 CXX - Moose の研究 XII CreateGMap を書き直す

オブジェクト指向モジュールをどのように書くのがよいか試したりしている。継承のサンプルを作りたいのだが、まだ適当な構成のイメージが浮かばない。

試したのは、出力するHTMLを一つのメソッドで出力できるようにするために、YAML形式の地域情報を読み込むための部分を CreateGMap.pmに取り込んで、gmapメソッド一つにまとめること。YAMLファイルのフルパスを変数として加える。このようにすると、ほとんど無意味で不自然なヘッダ出力、ボディ出力、フッタ出力のメソッド分割はなくなる。HTMLテンプレートとパラメータを載せた大きな一つのコンテナがで きあがる。本体のスクリプトからは、CreateGMapだけの使用を宣言し、コンストラクタを作ってメソッドを一つ呼ぶだけで済むようになる。

```
package CreateGMap;
use Moose;
use CreateGMap::Area;
use YAML qw(LoadFile);

has 'baseurl' => (is => 'rw', isa => 'Str', default => '');# マーカーデータ中の相対URL補完用
has 'key' => (is => 'rw', isa => 'Str', default => '');# Google Maps KeyのURL文字列
has 'title' => (is => 'rw', isa => 'Str', default => '');# ページタイトル
has 'desc' => (is => 'rw', isa => 'Str', default => '');# ページの説明
has 'id' => (is => 'rw', isa => 'Str', default => 'map');# Google Mapの表示位置
has 'width' => (is => 'rw', isa => 'Int', default => 500);# Google Mapの幅
has 'height' => (is => 'rw', isa => 'Int', default => 600);# Google Mapの高さ
has 'messageid' => (is => 'rw', isa => 'Str', default => 'message');# Google Mapのセンター経度緯度表示位置
has 'lng' => (is => 'rw', isa => 'Num', default => 132.472833);# 経度
has 'lat' => (is => 'rw', isa => 'Num', default => 34.385555);# 緯度
has 'data' => (is => 'rw', isa => 'Str', default => 'data.xml');# マーカーデータファイル
has 'zoom' => (is => 'rw', isa => 'Int', default => 1);# Google Mapの縮尺設定
has 'areafile' => (is => 'rw',
                   isa => 'Str',
                   default => 'C:/full/path/area.txt');# YAML形式地域情報ファイル

sub gmap {
        my $self = shift;
        my $html = <<HEADER;
Content-type: text/html; charset=UTF-8

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-microsoft-com:vml">
<head>
.....
</head>
HEADER

        $html .= <<BODY;
<body onload="onLoad($self->{lng}, $self->{lat}, '$self->{data}', $self->{zoom})">
```

```
<h3>$self->{title}</h3>
<p>$self->{desc}</p>
<hr>
<table border=0>
<tr>
<td>
<div id="$self->{id}" style="width: $self->{width}px; height: $self->{height}px"></div>
<div id="$self->{messageid}"></div>
</td>
<td valign=top>
        <table>
BODY

### 地域情報の YAML ファイル読み込み
        my @hashref = LoadFile($self->{areafile});
        foreach (@hashref){
                my $area = CreateGMap::Area->new(%$_);
                $html .= $area->tableunit;
        }
###

        $html .= <<FOOTER;
        </table>
</td>
</tr>
</table>
.....
</body>
</html>
FOOTER

        return $html;
}

1;
```

CreateGMap::Area モジュールは前と同じものを使う(言葉と物 CIX - Moose の研究 I まえがきなど)。本体のスクリプトは次の様に簡単になる。

```
#!/Perl5.8/bin/perl.exe
use CreateGMap;

my $gmap = CreateGMap->new(
        baseurl => '<- リンク URL が相対的な場合に他のサイトに表示する場合に使う ->',
        key => '<- Your Google Maps Key ->',
        title => '<- Page Title ->',
        desc => '<- Page description ->',
        areafile => 'C:/full/path/area.txt'
```

```
);
print $gmap->gmap;
```

このサンプルはPerlスクリプトとしては少し単純すぎるが、複雑な要素の大部分は、モジュールに書き込まれたHTMLテンプレートの中に含まれて いる。HTMLのメタデータ要素、CSS、Google Maps用のJavaScriptコード、Ajaxによる表示用のためのHTML要素、tableタグによる地図と切替え用地域情報の配置、コピーライト、Google Analytics用のJavaScriptコードなど。テンプレート部分だけでも3種類の人工言語が自然言語中に埋め込まれており、変数はPerlで記述されている。テンプレートの構成要素をさらに細かく制御できるようにしようとすると、また別の言語を生み出しかねないのではあるが、他の人に使ってもら うためには、ワンパターンを受け入れるのであれば、もう少しパラメータをテンプレート用に付け加えておくぐらいでよいだろう。

このようにモジュール化を考えてみると、HTMLの構造が複雑化しているのがよくわかる。Google Map部分の表現はJavaScriptが大部分を占めているし、地図のマーカーデータは、XML形式で読み込まれる。HTML、JavaScript、 XML、CSS、Perl、YAML、そして自然言語を含めて、なんと7種類の言語を使用する複雑なハイブリッド言語システムとなっているのだ。もっともYAMLは地図の地域の切り替えデータを導入するために、Perlは動的なHTMLの生成のために、僕が付け加えた言語である。YAMLはXMLか JSONでもよいだろうし、Perlの部分はJavaScriptに置き換えられるかもしれない。

なぜ言語システムは複雑化するのか。世界を表現するための多様な表現力を獲得するためである。Perl 4について書かれたPerlプログラミングの初版赤本(1993年)は633ページ、Perl 5について書かれた第2版の青本(1997年)は759ページだったが、Perl 5.6について書かれた第3版(2002年)は、2分冊になり計1303ページとなった。何が増えたのか、まったく新しいものはUnicodeぐらいだろうが、スレッド、セキュリティ、CPAN、プラグマモジュールなど、大きな項目になっているものも多い。関数や正規表現など言語仕様も拡大したのだろう が、時代とともにライブラリが増え、ドキュメンテーションが進んだことも大きな理由だろう。内容が豊富になれば、当然分量が増えるが、それは一貫して自然言語で記述される(一部はPerlコードであり、原書は英語であるわけだが・・・)。しかし、このお話しは人間の暗黙知の共有化における自然言語上のプロセスに過ぎず、コンピュータ上の言語システムの複雑化とは関係ない。CreateGMapシステムは、コンピュータ上に世界を精緻に表現するための遠い道程の道草なのか、一時的な気の迷いなのか、定かではない。

昔は^^;)、万物は原子からなると考えられたが、今でもものごとの基本的な要素をatomと表現することも多い。それはさらに下部の構造を持っており、多様な素粒子の存在が検出されるようになる。さらにクォークが現われる。現在では超ひも、そしてDブレーン(brane;膜)が想定されるように なってきている。今のところ、コンピュータの基本はビットである。機械の上にどのような時空が構成できるのか、オブジェクト指向を廻る旅はまだまだ続くだ ろう。

更新: 2008-03-29T23:37:06+09:00

3/24/2008 (Mon.)

[オブジェクト指向] 言葉と物 CXXI - Mooseの研究 XIII 特性と状態と糊

サブルーチンのパラメータという言い方は、関数指向的な表現だと思う。関数なら変数(パラメータ)の値を入力するとサブルーチンに記された計算式に 基づいて計算結果を出力するというイメージとなる。クラス(Perlならパッケージ)は関数(サブルーチン)の集まりのようにも考えられる。どこで、クラ スがオブジェクトを生み出すようになったのか。

本日のタイトルは「特性と状態と糊」、この三つの要素を足すとクラスが出来上がるという話。Class::Traitの引用文献の類似文献として だったと思うが、「Classes = Traits + States + Glue - Beyond mixins and multiple inheritance」(Mathanael Schaerli et al,The Inheritance Workshop

at ECOOP 2002)を CiteSeer で見つけた。クラスの特性の集合から必要な状態を得るための糊が、すなわちメソッドである。

そうすると、traits とは何かということになるわけだが、property や attribute と同じように見える。traits の意味はもう少し調べてみよう。Moose はクラスの属性にメタクラスの属性(Class::MOP::Attribute)に加えて、Moose::Meta:: Attribute の属性を指定できる。これまでの Moose を使ったパッケージサンプルでは、普通のクラス(パッケージ)の new メソッドを定義する代 わりに、has メソッドでクラス属性を定義する。クラス属性のアクセス定義、型定義、デフォルト定義などが可能である。特に型定義の属性は、コンテキストによって型の変化するダイナミックな言語においては、型が変化しては困る場合には、プログラムの一貫性を保つために重要なのだろう。

更新: 2008-03-29T19:12:54+09:00

3/29/2008 (Sat.)

[オブジェクト指向] 言葉と物 CXXII - Moose の研究 XIV 古い方法との比較

最先端から少し過去に遡って古い書き方と比較してみよう。まずは、実践実用 Perl のオブジェクト指向 Perl モジュールミニ入門の原始的レベルまで戻る。おそらく、当時は perltoot のサンプルを参考に書いたはずだ。

```
package CreateGMap;
use CreateGMap::Area;
use YAML qw(LoadFile);

sub new {
        my $self = {};
        $self->{BASEURL} = undef;
        $self->{KEY} = undef;
        $self->{TITLE} = undef;
        $self->{DESC} = undef;
        $self->{ID} = undef;
        $self->{WIDTH} = undef;
        $self->{HEIGHT} = undef;
        $self->{MESSAGEID} = undef;
        $self->{LNG} = undef;
        $self->{LAT} = undef;
        $self->{DATA} = undef;
        $self->{ZOOM} = undef;
        $self->{AREAFILE} = undef;
        bless($self);
        return $self;
}

sub baseurl {
        my $self = shift;
        if(@_){ $self->{BASEURL} = shift };
        return $self->{BASEURL};
}
```

```
.....
sub areafile {
        my $self = shift;
        if(@_){ $self->{AREAFILE} = shift };
        return $self->{AREAFILE};
}

sub gmap {
        my $self = shift;
        my $html = <<HEADER;
Content-type: text/html; charset=UTF-8

.....
<head>
.....
        <title>$self->{TITLE}</title>
.....
}

1;
```

起動するには、これまでのサンプルとは少し違う。

```
use strict;
use warnings;
use CreateGMap;
use CreateGMap::Area;
use YAML qw(LoadFile);

my $gmap = CreateGMap->new();
$gmap->baseurl('<- リンクURLが相対的な場合に他のサイトに表示する場合に使う ->');
$gmap->key('<- Your Google Maps Key ->');
$gmap->title('<- Page Title ->');
$gmap->desc('<- Page description ->');
$gmap->id('map');
$gmap->width(500);
$gmap->height(600);
$gmap->messageid('message');
$gmap->lng(132.472833);
$gmap->lat(34.385555);
$gmap->data('data.xml');
$gmap->areafile('C:/full/path/area.txt');
$gmap->zoom(1);

print $gmap->gmap;
```

この方法でも起動時にオーバーライドできるので、undef の代わりにデフォルト値を設定しておくこ

ともできる。

プログラミング Perl 第3版(オライリー・ジャパン、2002年)の373ページ(Vol.1)に紹介されている方法(12.4.2 イニシャライザ)を使ってみよう。

```
package CreateGMap;
use CreateGMap::Area;
use YAML qw(LoadFile);

sub new {
        my $invocant = shift;
        my $class = ref($invocant) || $invocant;
        my $self = {
                baseurl => '<- リンク URL が相対的な場合に他のサイトに表示する場合に使う ->',
                key => '<- Your Google Maps Key ->',
                title => undef,
                desc => undef,
                id => 'map',
                width => 500,
                height => 600,
                messageid => 'message',
                lng => undef,
                lat => undef,
                data => 'data.xml',
                zoom => 1,
                areafile => 'C:/full/path/area.txt',
                @_,
        };
        return bless $self, $class;
}

sub gmap {
.....Moose のサンプルの gmap メソッドと同じ
}

1;
```

起動の仕方はMooseのサンプルと同じ。この方法でも通常の利用には機能的には十分だろうと思うが、なぜそのように書く必要があるのか、わかりにくい。Mooseは非常にスマートで、直感的にわかりやすいし、明示的である。

更新: 2008-04-01T22:11:55+09:00

3/30/2008 (Sun.)

[オブジェクト指向] 言葉と物 CXXIII - Mooseの研究 XV 知識表現としてのオブジェクト指向

オブジェクト指向の目的がコードの再利用や大規模プログラミングの効率化であることは事実では

あるのだろうが、その程度のことなら、「言葉と物」シ リーズで僕が取り上げるほどのことでもない話である。プログラミングは、特にテキスト処理は知識表現の一種であると考えてよい。コンピュータとネットワー クは知識が蓄積され、かつ表現される場なのである。

知識の複雑な構造を表現する方法は進歩し、多様化してきている。昔ながらの書物はWikipediaのようにハイパーテキストで書かれたWebペー ジに変貌するのだろうか。ワードプロセッサのファイル形式はXML形式で標準化されている。PDFのXML表現はどうなっているんだろう。テキスト、表、 図、グラフ、画像、写真、ビデオ、そしてハイパーリンク、データベース。今やWebでは何でも取り扱える。Webにおける知識表現を考えることは言うまでもなく重要なことである。

さて、タイトルの答えは、永遠の課題でもあるだろう。さらなる探索と思索の旅を続けよう。

- 知識表現 - Wikipedia
- Knowledge representation

Wikipediaの知識表現のページは簡潔によくまとまっている。オブジェクト指向プログラミングで使われる用語は知識表現の用語と重なる部分も多い。「has」、「is-a」、「slot」などの用語は「知識フレーム」から来ている。オントロジーエディタのProtégéを見ていると、slotという入力項目があるのだが、その出所がわかった。

情報処理学会電子図書館で、会誌を「知識表現」で検索してみると55件がヒットする。オブジェクト指向に関わる記事として、溝口文雄氏の「オブジェ クト指向概念による知識表現言語」(情報処理、Vol.29、No.4、Apr. 1988)という解説を見つけた。知識表現はマーヴィン・ミンスキーのフレーム理論(1975年)に基礎を置いている。

更新: 2008-04-06T08:29:26+09:00

4/1/2008 (Tue.)

[オブジェクト指向] 言葉と物 CXXIV - Mooseの研究 XVI 知識表現言語

月が変わってしまったので、ページを変えよう(言葉と物 CXXIII - Mooseの研究 XV 知識表現としてのオブジェクト指向)。 知識表現するための知識にも様々な領域がある。まず表現対象に関する知識、そして対象を表現する方法についての知識、大きく分けて二つある。コンピュータ上のプログラムにおいては、例えば、表現対象のデータ、データの記憶・加工・表示に関する手続きと操作方法などのインターフェースを記述する必要がある。

「オブジェクト指向概念による知識表現言語」の最初には、KRL(Knowledge Representation Language、中途で開発が中止される)と一緒にMark Stefik氏のUnit Packageが紹介されている。ここにSlotの概念が現われる。詳細には、IJCAI-79(第6回人工知能国際会議、東京、1979年)で発表された「AN EXAMINATION OF A FRAME-STRUCTURED REPRESENTATION SYSTEM」を見るべきだが。これを読むと、フレーム構造表現といっても意味ネットワークの概念が色濃いことがわかる。「2 表現の要素」の項の最初には「Unit Packageの知識は区切られた意味的ネットワークとして組織される。KRLの用語法に従うと、nodeがunitと呼ばれ、linkはslotと呼ば れる。組み込みの一般化関係性はいくつかのモードの特性継承によって階層化をもたらす。」とある。インスタンス(instance)、継承(inheritance)、制約(constraints)、methodに通ずる付属手続き(attached procedure)のようなオブジェクト指向プログラミングで使用される用語やaspectの概念が登場し、SIMULA(1966)やSMALLTALK(1983)のプログラミング言語の名前は見えるが、「object」の言葉は、まだない。

オブジェクト指向プログラミングと知識表現言語との一致を宣言するのは、LOOPS(Lisp Object-Oriented Programming System)を作ったStefikとBobrow(The LOOPS Manual, 1983.)の「Object-Oriented Programming: Themes and Variations」(Mark J. Stefik and Daniel G. Bobrow, AI Magazine, 6(4):40-62, 1986.)を待たねばならない。最初、オブジェクトに結びついたのはWindowシ

ステムであり、知識を表現する方法は正に知識表現だったのである。初 期のテキスト処理がワードプロセシングそのものだったのと同じことである。

松岡正剛の千夜千冊を久しぶりに覗くと、1230 夜(2008-03-25)の『一般システム思考入門』ジェラルド・ワインバーグに node と link が出てきたので驚いた。node と link から Semantic network が気になる。思わぬところで、Sowa の意味の三角形に再接近。XML や RDF も意味ネットワークであり、フレームでありということか・・・

- オントロジー、メタデータ、そして記号学 (2004/10/27)
- 意味と実体 (2005/01/09)

更新: 2008-04-03T23:12:16+09:00

4/6/2008 (Sun.)

[言語] 言葉と物 CXXV - Moose の研究 XVII 言葉で超える

難波江和英、内田樹著、「現代思想のパフォーマンス」(光文社新書、2004 年;原著: 松柏社、2000 年)。現代思想ネタ。パフォーマンスという言葉に惹かれたのだが、思想も役立たなければ無意味だ。まえがき、あとがきと最初のほうを齧っ てみただけだが、難解な現代思想の使い方が書かれているらしい。それはともかく、最初の一人がソシュール。

おそらく小学生ぐらいの時のことだが、その頃は「犬」のスピッツという白くて毛がふさふさした中型の種類が愛玩犬として人気があった。テレビでは 「名犬リンチンチン」のシェパードや「名犬ラッシー」のコリーが有名となり、子供心には魅力的に映っていた。しかし、身近に見る雑種の飼い犬や捨て犬の様 々な多様な形態の「犬」を見ながら、これは全部「犬」なのだろうか、なぜ「犬」だと思うのだろうかと不思議に思った覚えがある。

「身分け」、「言分け」という言葉がある。丸山圭三郎氏の用語で、知覚による分類(大袈裟に言えば世界認識)と言葉による分類を言う。難波江氏の解説によれば、「身分け」は「言分け」によって限界づけられているというところに引っ掛かった。言葉で世界が構造化されているからというわけだが、あまりに も静的な見方と思うからだ。人間は新しい言葉をいくらでも生み出すことができる(言語の変遷、言語の数を見れば明らかである)。あるいは他の言語を学習す ることもできる。機械用の言語を生み出し、操ることさえできる。逆であって、「言分け」は「身分け」で限界づけられているのが真相であろう。もっとも「身分け」は「言分け」によって構造化され続けるので認識能力を高めているとも言える。「身分け」に認知の基本はあるが、「言分け」は創造的認知能力であり、 限界づけられるというよりは、限界を超えるためのものである。

- 意味野の構造 (2004/02/29)
- 脳からみた心 (2004/03/07)
- 意味と実体 (2005/01/09)
- 言葉と概念 (2006/10/11)
- 言葉と物 ＸＸＸＸＸⅡ - 哲学と科学のあいだ (2007/09/02)
- 言葉と物 LVIII - ソシュール (2007/09/15)

いろいろと自らの過去の言説をチェックしていて、今年の一月に Semantic Wiki について で次の様に書いているのを見つけた。

> ・・・XML/RDF はデータの構造を考え抜いて設計するわけだが、新しい要素を後から付け加えたくなる場合もあるだろう。人間の思考とはそのよう なものである。オブジェクト指向の用語で言えば、継承ということになるのだろうけど、自己組織化であるためには、相互継承というか、オブジェクト間で相互に連想が生じるような仕組みが必要かもしれな

い。自然にデータが生成して、まとまる部分は自然にまとまるというか・・・そんなイメージ。

また、Frame Languageの「Inference and reasoning」の項には次の様に記載されている。

> フレームベースの知識表現において、継承(inheritance)は推論(inference)の中心的な機構となっている。フレームは、根(ルート)フレーム」としての一般的な概念を持つ階層(hierarchy)として組織化される。多くのシステムは多重継承をサポートする。これらのシステムにおいて、木構造は可能なサイクルを持つ有向グラフ(directed graph)のように見ることができる。
>
> フレームシステムの論証(Reasoning)は、フレーム・マッチング、継承と活性伝播(spreading activation)に基礎を置いている。

取りあえずここでは、継承というのは推論と関係があるということを指摘するのに留めておこう。活性伝播というのはわかったようでわからないような訳だが、意味ネットワークなどを探索する方法を指している。

更新: 2008-04-06T12:59:03+09:00

4/11/2008 (Fri.)

[言語] 言葉と物 CXXVI - Mooseの研究 XVIII メタ言語と神話

オブジェクト指向の研究は、思いがけず、フレーム理論、そして意味ネットワークへと通じる道を辿った。起源まで追いかければ、紀元前まで遡るらしい。単なる手続き的なプログラミングをいくらオブジェクト指向で書いても、継承になぞ、なかなかお目にかかれない。しかし、フレーム理論や意味ネットワー クでは継承は必然だ。さて、何を継承するのか・・・

Moose::Objectが提供するオブジェクトは一種のファイルシステムのようなものだ。変数などの型定義や読み取り属性定義などの機能が提供 される。変数の持つ一次的な意味とは意味的な階層が異なる。メタオブジェクトなのである。ファイルシステムのようなものはむしろソフトウェアの基盤であって、アプリケーションそのものの意味的な上位にあるというよりは下位にあって支えるものである。その変数の持つ属性を継承するので、継承する方向は決まっている。

「現代思想のパフォーマンス」のロラン・バルトの章を読んでいると、神話は二次的な記号システムであるという話が出ていて、一次的な語義どおりのシ ニフィエとシニフィアンを示すシーニュは二次的なシニフィアンとなり、次元の異なるシニフィエを持つシーニュを生み出す。これが神話である。ランガー ジュ・オブジェ(素材としての言語)のシーニュをメタ・ランガージュ(メタ言語、上位言語)のシニフィアンとして継承する。継承の方向が逆かな、自然言語 の継承は。しかし、元の語の意味と響きあう関係にあり、相互継承的な関係にあるとも思える。うむ。ちょっと違うな・・・図版8はコノテーションを示していると思う。メタランガージュは意味的なものの継承になるので、シニフィエ(signifié → Sé)に継承する。コノテーションは外形を継承するので、シニフィアン(signifiant → Sa)に継承する。

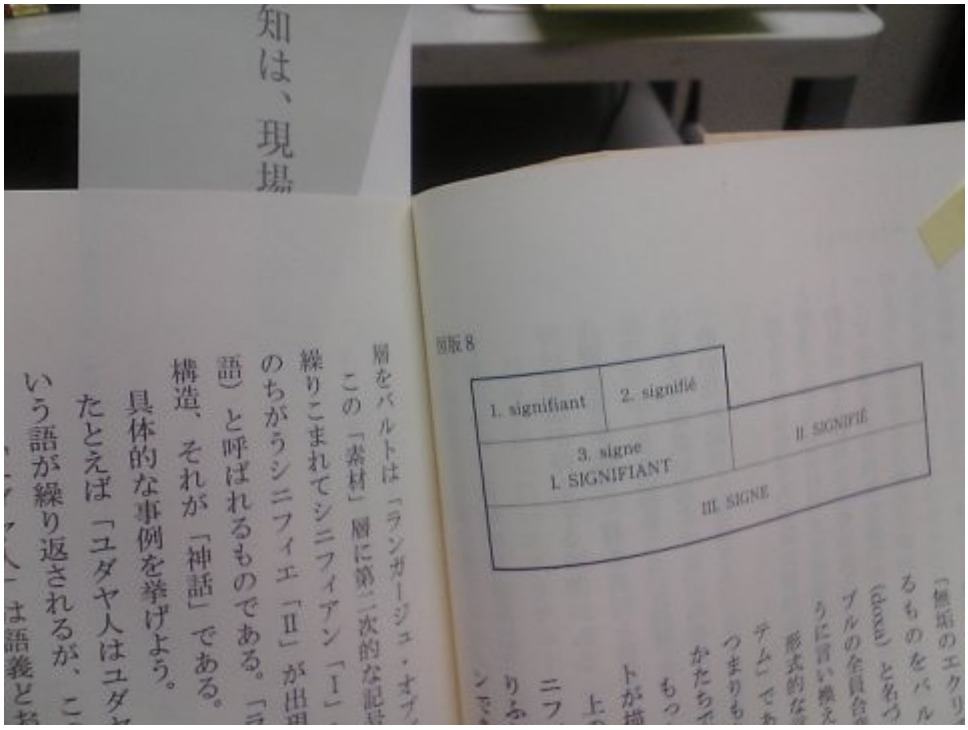

図.　神話的イデオロギーの構造

この図(図版8)は意味的には上下が逆であるべきだと思う。また、この図はメタランガージュではなく、コノテーションを示している。(「現代思想のパフォーマンス」、光文社新書、2004年、102-103ページ)

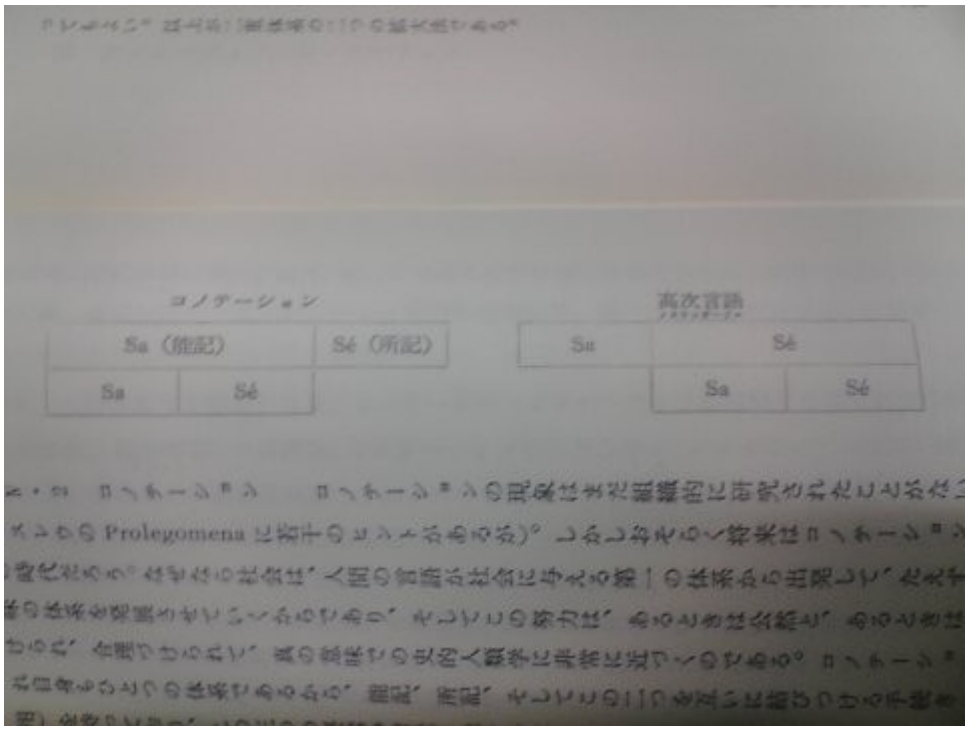

図.　コノテーションと高次言語(メタランガージュ)

コノテーション的体系とは外形の面がひとつの記号作用体系によって作られた体系である。メタランガージュとは、内容の面がひとつの記号作用体系で作られている体系である。観念形態(イデオロギー)とは結局、コノテーションの所記(シニフィエ)の形式である。(ロラン・バルト著、渡辺淳、沢村昂一訳「零度のエクリチュール 付・記号学の原理」、みすず書房、1971 年、196-198 ページ)

- 「言葉と物」ＸＸＶⅡ － 二次的な意味 (2007/06/23)
- 言葉と物 ＸＸＸＸⅢ － 意味を追うソフトウェアシステム (2007/07/29)

言葉の継承関係は複雑だ。言葉の語義自体も多態的であり、様々な意味があり、語尾が変化せずとも品詞的に多態である場合もある。言葉の使用におい て、文、段落、節、章、作品、作品群、等々、様々なレベルがある。その継承関係の解析は気が遠くなるようなものだろう。結局、単に解析したところで意味が それほどあるわけではない。むしろ、新たに生み出すことのほうが意味的な継承関係を成立させるには効率がよい。それが創造の存在理由なのである。

更新: 2008-04-13T21:02:04+09:00

4/17/2008 (Thu.)

[W3C] 言葉と物 CXXVII - Common Web Language

言語バリア撤廃へ向けて、最終報告書Common Web Languageネタ。これは最高におもしろそう。

- Common Web Language

CWL の目的は、言語の障壁を超えて、メタデータだけでなく内容も記述できるようにすること。メタデータはその部分に書かれている内容のカテゴリ (名前空間)を指し示すだけだ。メタデータからは書かれている内容自体はわからない。言語の障壁を超えて、内容自体をわかるようにすることってどういうこ とだろう。結局、それは世界共通言語を作ろうとする試みであるはずだろう。機械も理解できるから、機械翻訳にも使えることになる。「1.2 Requirement for CWL」の部分を訳してみよう。

1.2 CWL の必要事項

上記の目的を達成するための CWL の必要事項は次の様になる:

1. 自然言語から独立であること、そして使用者が CWL と各々の自然言語の間の変換システムを容易に開発できるようにすること。
2. 自然言語と違い、CWL は一義的な形式言語(unambiguous formal language)であり、人間にとっての自然言語と同じ役割を演ずる。
3. 自然言語に含まれる概念(concepts)は、CWL の語彙であることができる。
4. 原文書の表面構造と意味構造が CWL で表現されなくてはならない。

実際に CWL で何か表現してみたいかということになると、新しい言語を修得するのと同じぐらい大変な作業になるはずだから、XML/RDF 以上に困難な感じはする。しかし、興味深い試みであることには変わりない。簡単にできることでは大したものにはならないからだ。

CWL において、概念は、言語の形態素(morpheme)、語、句、あるいは文と UNDL Foundation で 開発された普遍語(Universal Word)で表現することができる。CWL の語彙、概念間の関係、属性の項を見ると、node と link の関係を思い出す。CWL は意味的ハイパー有向グ ラフ(semantic hyper directed graph)のグラフ言語であり、ノードは概念を表し、弧(arc)はノード間の関係を表し、ノードは属性に

よって注釈される。

CWL を表現するために、UNL、CDL、RDF が使われる。

- UNL Specifications Version 2005, Edition 2006
- Institute of Semantic Computing(Concept Description Language)

更新: 2008-04-19T21:58:50+09:00

4/25/2008 (Fri.)

[言語] 言葉と物 CXXVIII - 日本語の特徴

日本語って変かもネタ。 この内田先生のブログを読んで、養老先生の文章を思い出した。先週だったか、ゆめタウンの紀伊国屋書店で目に入った養老先生の本は、持っていないとすぐわかったので3冊購入。文春文庫の「涼しい脳味噌」(1995)、「続・涼しい脳味噌」(1998)の2冊と新潮文庫の「身体の文学史」(2001)。最初 の2冊は1989-1994年ぐらいに雑誌に書かれたものを集めたものだ。「身体の文学史」は1997年に単行本が出ている。「涼しい脳味噌」の中に「日 本語の特徴」という文章がある。

日本語の特徴は漢字かな混じり文にあるという話でそれはそうだねと思ったのだが、漢字に音訓と複数の読み方があるのは、かなりおかしいことになって いるという指摘になるほどと感心した。ソシュールが言うには、言葉のようなシーニュ(記号)はシニフィアン(能記)とシニフィエ(所記)の二面性を持つ。 シニフィエは概念、シニフィアンは聴覚映像とも訳される。日本語の場合、漢字が使われる場所によって、読みが変化する。シニフィアンは聴覚映像だけでなく、視覚映像でもあるわけだ。従って、日本人は脳の二箇所を使って言葉を認識することになる。

日本語が単なる漢字かな混じり文であるだけでなく、アルファベットや他の言語記号を取り込んでもほとんど違和感がないように感じるのも、視覚的に言葉を捉えているからかもしれない。

連休に入るので昨年の5月に書いた記事を読み直していると、05/16/2007:[養老孟司] 言葉と物 XV - 唯脳論が 目に留まった。「涼しい脳味噌」の解説を読んでいると養老先生の最初の著作は「ヒトの見方」(1985)なのだそうだ。「形を読む」、「唯脳論」 (1989)はその後に書かれた。「ヒトの見方」を書く動機は「ただ、いわゆる自然科学の文章を、日本語で表記したいという気持ちが、自分の大きな動機 だったことは間違いない」そうである。「ヒトの見方」の最後の「鴎外とケストラー」に「自然科学でもっとも重要なものは、形式である。方法はこれに含まれ る。科学論文は、一定の手続きを踏んで行われた仕事を、一定の手続きで表現しなくては、ふつう認知されない。・・・」(ちくま文庫、337ページ)とあ る。形式をフレーム、方法をメソッドと読めば、自然科学とは知識表現であると当然の結論に至るのである。

「ヒトの見方」の文庫版のあとがきには次の様にある。

> 言語の問題は、『唯脳論』の中で、もっと発展したと思う。「日本語で書く」だけでなく、「日本語を使う」ことの意味が、すこしずつ、分かってきたからである。・・・

更新: 2008-04-28T23:37:17+09:00

4/28/2008 (Mon.)

[プログラミング言語] 言葉と物 CXXIX - プログラミング言語の意味論

デイナ・ スコット - Wikipedia経由、領域理論ネタ。

今日、渕先生の「機械の上の意味論」(竹内啓編、「シリーズ・人間と文化●2 意味と情報」、東京大学出版会、1988 年、55-72 ページ)を読み直していて、指示的意味論の項で、次のような部分に出会った。それで、調べた。

指示的意味論

もう一つの方法は、伝統的な記号論のセマンティックスの考えに沿うもので、数学的論理学でのモデル理論に相当するものです。指示物との対応関係を与えることによって、意味を確定しようとする立場です。

数学の世界の中にモデルを作って、それとの対応を与える。論理学の場合は、集合論を利用してモデル理論を作っています。これを右から左に流用できれば楽だったのでしょうが、プログラミング言語の場合、普通の集合論では問題があったようです。そこでスコットの新しい集合論モデルなどが現れた。

(意味と情報、64-65 ページ)

指示的意味論という言葉は最近は使われていないようで、Wikipedia では表示的意味論となっているようだ。表示的意味論は denotational semantics の訳であり、denotation の訳には、表示という意味もあるようだが、辞書的にも明示的意味、指し示すもの、指し示すことのような意味が挙げられており、表示という静的な訳語は少し弱い感じがする。ところで、denotaion はロラン・バルトの「デノテーションとコノテーショ ン」のデノテーションに対応するわけで、デノテーションは明示のためにメタ言語化していく。これがオブジェクト指向の原点なのだろう。渕先生の「機械の上の意味論」には自然言語の意味論についても別の項目を立てて紹介されている。

自然言語の意味論には次のような記述がある。

手続き的意味論

・・・・・

ところで、十九世紀にパースという哲学者がいました。彼はプラグマティズムの提唱者であり、記号論の創始者の一人であるのですが、彼は、記号の意味というものをその記号が引き起こす「効果」としてとらえようとしました。

記号が効果を持つといっても、記号が直接働くわけではなく、記号を受けて何かの効果を発揮するなんらかの仕掛けが途中に介在するという二重構造に なっているのが普通です。それがインタープリタです。ところで、インタープリタを抽象的なコンピュータとして設定しようというのが操作意味論であったわけ です。

そのインタープリタも、基本となるコンピュータを決めれば、ソフトウェアすなわちプログラムの形でそれを分離することができます。とすれば、そのプログラムが意味であると言ってよいわけです。

(意味と情報、67-68 ページ)

プログラムが意味とすれば、プログラムがオブジェクト指向化するのは必然的な道筋なのだろう。原稿の締め切りはとっくに過ぎ、時間切れ、ここをオブジェクトを廻る日記の一つの区切りとしよう。

更新: 2008-04-28T20:43:45+09:00

Afterword

本稿の当初の構想は、書き綴った日記を並べた上で統合し、整理したものになるはずであったが、その計画は日記を書くのに要した時間以上の時間を要することになることに気が付いて、あっさりと放棄した。また、日記を書いた時点の理解不足のために間違いも多いかも知れぬが、敢えて、新たに修正や変更も行わなかった。

本稿の最初の目的は、Perl 5 の Moose というオブジェクト指向システムモジュールを使用する具体的なアプリケーションを作り出すことであった。その目的は早々と達成できたのだが、それだけではまったく満足できなかった。

知識の連鎖を辿る更なる探索を元に結論を何か書ける段階まで、思索は深まっていない。まだオブジェクトの表面をたどたどしく撫でているだけである。Perl 5 で Moose を使えば、簡単にオブジェクト指向モジュールを作れることはわかったが、それを活用して手続き指向のプログラミングに対比して、具体的にレベルの違う何かができるのかどうかはまだよくわからない。具体的なアプリケーションにおいてオブジェクト指向の意味が今ひとつ見えてこなかったからである。

Perl 6 のオブジェクト指向プログラミングで構想されている ROLE や Traits は、Perl 5 の Moose で試すことができる。今後の課題は、Unit Package から得られるインスピレーションをもとに、自らのオブジェクトをさらに構造化してみることだろう。

今後、オブジェクト指向の概念はまだまだ変化していくのではないかと思う。現在のオブジェクトは他のオブジェクトとの相互作用という観点から考えると、継承という性質はあるが静的なものである。オブジェクトとオブジェクトの相互作用のようなものを記述できるようになればおもしろいのかもしれない。

(2008-05-05)

編集後記

閑舎

ss.pl や mylib.pl、こんなシンプルなスクリプトでも自分のローカルマシンかレンタルサーバ上で CGI をかぶせて動かせば、もう立派な Wiki 風 CMS となるでしょう。

私はそうやって PukiWiki がサポートする大半の文法規則を HTML ファイルに変換するスクリプトを動かしています。

この種の整形ツールについての需要は昔からあって、いわゆるフォーマッタと呼ばれています。

HTML も XML も TeX もタグ付けされたテキストという意味で、こうしたフォーマッタの素材としてもってこいです。

Wiki 風スクリプトのもう少し高度な議論は TSNET で交わされています。ご興味があれば是非ご参加を。

---

jscripter

さて、ようやく編集作業も終わりに近づいてきた。本会誌は、OpenOffice.org を使わせてもらって、編集している。素晴らしいツールである。このようなツールがあるおかげで、アマチュアが電子出版を容易に行うことができる。スクリプティング言語などを含め、有用なコンピューティングツールがフリーかつオープンに提供されるようになったことは大変ありがたいことである。フリーソフトウェアの関係者に大変感謝したい。

電子出版のメリットはハイパーリンクをドキュメントに残せることである。自分自身にとっても大変有用なまとめができたと喜んでいる。

TSNET にさらに多くの人に参加していただいて、TSNET スクリプト通信を年に数回程度、発刊できるようになれば素晴らしいと思う。是非、ご参加を^^)/~

Ｙさ

...始まりはいつもゲームで。

貴方をテキスト&スクリプトに相応しいプログラミングの世界へ誘います、なんちて^^;)

今回ゲームを書きました。人と対話的に情報のやり取りをするプログラミングを学ぶ題材として向いてると思っているわけなんですよ、ゲームは。　「(コンピュータゲームの進行は)プレイヤーの入力に対する結果をコンピュータが演算し、その処理結果に対してさらにプレイヤーが次の入力を行うという繰り返しによってなされる」と『ウィキペディア』にもありますし。

ところで「貧相な画面のゲーム第一弾」が気になった方、TSNET スクリプト初心者の部屋(TSabc)を訪ねると幸せになれるかもしれません。(現在ひっそりと w 公開中)

p.s. 「GAME」と言えば Perfume です。武道館ライブチケット取れるかが目下心配の種...

機械伯爵

情報の授業を、これで４年くらいやっているのですが、気になるのは生徒のパソコンに対する考え方の変化。パソコンが生活の中に入るようになってから「パソコン嫌い」の子が徐々に増えつつあるような気がします。コンピュータが夢の機械だった私の子供時代を考えると、ジェネレーションギャップが甚だしくて当惑することしきり。もっと楽しくコンピュータに向き合えるよう、これからも授業を工夫してかなくちゃなぁ、と思います。

ＴＳＮＥＴスクリプト通信

2008 年 5 月 13 日　　1.1.001 版発行 PDF 情報改訂
2008 年 5 月 7 日　　1.1.000 版発行

投稿規程

TSNETWiki ： 「投稿規程」のページを参照のこと（URL は下記の通り）

編集委員会(投稿順)

Ｙさ	saw at mf-nokuchi2pho dot ne dot jp
閑舎	rakunet at gmail dot com
機械伯爵	kikwai at livedoor dot com
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 2.0.4, 2.4.0, 2.3.1 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
