TSNETスクリプト通信

TSC編集委員会

ISSN 1884-2798

目次

巻頭言

jscripter

マスコミは事業仕分けの話題で盛り上がっていますが、景気は元気がありません。TSNETスクリプト通信第7号、第2巻第3号は21世紀の明るい未来を照らし出すでしょうか。

「TSNETスクリプト通信」は国立国会図書館の収集書誌部(ISSN日本センター)から国際標準逐次刊行物番号（International Standard Serial Number）を11月6日に取得いたしましたので、表紙と後付に記載いたしました。これで、形式的には国際的に認知された逐次刊行物となりましたので、さらに公報活動を進めていこうと考えています。読者のみなさんのご協力をよろしくお願いいたします。

本号は、ムムリク(きしだあつし)さんと稲葉準さんという二人の新しい執筆者を迎え、大変嬉しい幸運を呼ぶ第7号となりました。

Yささんは、例によってAWKスクリプト第7弾、配置された爆弾を高度な推理力と勘で除去する新作ゲームです。四半世紀も昔の雑誌に掲載されていたゲームへのオマージュとなっています。

海鳥さんの「よしおさんとロボ太」は作品の原点に戻って、ロボ太が命名されるまでの6点を転載させていただくことにしました。お楽しみください。

ムムリク(きしだあつし)さんから「眠り姫」を投稿いただきました。TSNETスクリプト通信初めての記念すべきSF作品です。加えて「Ruby/Tk をしてみませんか」を投稿していただきました。Rubyの記事は創刊以来二つ目ですね。

私は「スクリプティング言語の周辺」を投稿しました。最近、いろいろと触ったりしたこと、考えたことを過去から遡ってまとめました。

稲葉さんには「Sjisソフトウェアってなあに?」を投稿いただきました。JPerlユーザーにとっては、Perl 5.8/5.10が活用できるようになる大変うれしい実用的なガイドです。

機械さんの「やｎでレPython ～λの使徒編～」。遂に第6回です。さて、読もう^^)

表紙の写真は、ムムリクさんから投稿していただき、十勝岳の写真に。いやー、寒そうですけど、きれいですね。25年前の11月下旬撮影とのこと。表紙写真投稿歓迎です。

次号、第8号は新年、2010年2月を予定しています。お楽しみに。投稿歓迎です。

(投稿受付: 2009年11月29日;改訂: 2009年12月12日)

MagneticRepulsion

written by Yさ

1. このゲームは

前作(memory=いわゆるカード配置を高度な記憶力と勘で推理するゲーム)に引き続き、貧相な画面のゲーム第七弾。今回は、配置された爆弾を高度な推理力と勘で除去するゲームです。

今回も新作なのですが、実は四半世紀も昔の雑誌に掲載されていたオリジナル'元ネタ'ゲームが存在しています。要するにインスパイアされてできたオマージュ/リスペクトあるいはパクリという事です。(スクリーンショットから推測するに細かなルール等が全く異なるものになっているとは思うのですが...)

2. 動作環境は

例によって、最近のawkならOKだと思います。ちなみに動作確認は、

GNU Awk 3.1.6, mawk 1.3.3 MBCS R27

で(WinXPのDOS窓で)行なっています。

3. 遊び方は

```
>gawk -f MagneticRepulsion.awk[Enter]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

等で起動するとゲームスタートです。

1)最初にn×nのゲーム盤サイズを決めてください。もっとも簡単なサイズ 4 から上限は 10 まで選択できます。サイズが大きい方がより難しくなります。(...多分)

サイズを決めると、爆弾が太枠の中に隠されて適当に配置されます。全ての爆弾を攻撃して破壊する のがゲームの目的です。破壊すべき残りの爆弾の数が'BOMB'として表示されます。

'MISSILE'が爆弾を<攻撃>できる数です。
'PROBE'が爆弾の位置を<探査>できる数です。

2)<攻撃>:縦A～,横1～の文字の組み合わせで座標指定入力して下さい。なお入力は英字数字の順でも数字英字の順のどちらでも構いません。英字も大文字小文字を区別しませ

ん。

例) a1[Enter]

爆弾の位置を攻撃した場合は ″><″ を表示します。(攻撃成功)
爆弾以外の位置だった場合は ″??″ を表示します。(攻撃失敗)

3)<探査>：爆弾は強力な反発磁力を発生させていて、探査装置の進行方向を変化させます。

これを利用して、上下方向、左右方向から探査装置を打ち込み、どこから出てくるかで爆弾の位置を推測することができます。なお反発磁力は爆弾が破壊された後も残ります。

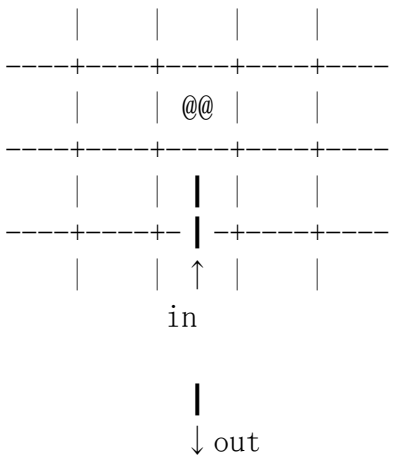
打ち込み位置は

上から下向き UA～,
下から上向き DA～,
左から右向き L1～,
右から左向き R1～

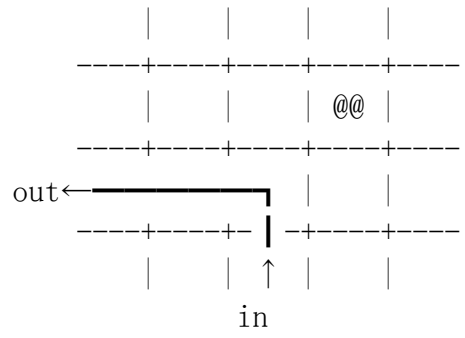
の英数字2文字で指定してください。英字の大文字小文字は区別しませんが、文字順は正しく指定してください。指定した2文字の文字列が、最終的な位置に表示されます。

以下、探査装置をinの位置から打ち込むと、爆弾の位置によってどこに出てくるかのパターン例を示します。″@@″ が爆弾の位置、in から打ち込むと out に打ち込み位置が表示される事を表します。

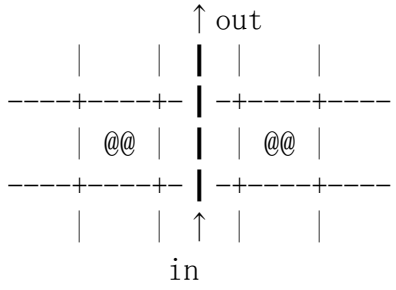
[パターン例1] 真っ直ぐ反射して戻る



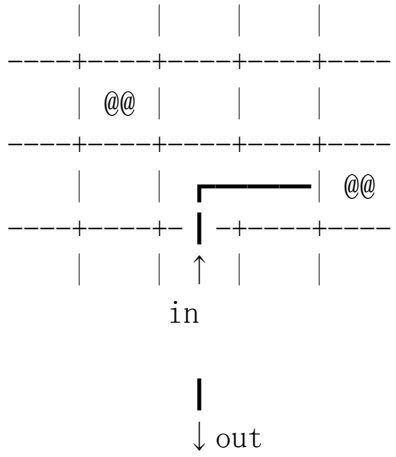
［パターン例 2］進行方向が直角に曲がる



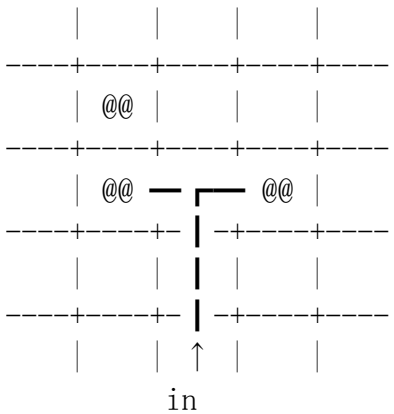
［パターン例 3］そのまま通り抜ける



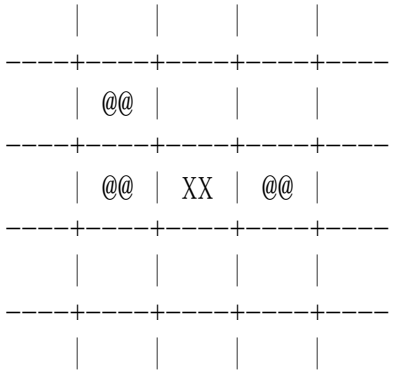
［組み合わせパターン例 1］曲がって反射して戻る



［組み合わせパターン例 2］堂々巡り(出てこない)



⇒ この場合は"XX"=打ち込み位置が盤の中に表示される



4)爆弾の配置によってはどうやっても'PROBE'を通せない位置が出てきますが、そこは「勘」で攻撃してください。

なお最後のぎりぎりで'MISSILE'が足りなくなる場合、'PROBE'が５つ以上残っていれば自動的にエネルギー変換され'MISSILE'が補充されます d(^^; ただし逆変換はできません。全ての爆弾の破壊に成功するか、'MISSILE'が 0 になるとゲーム終了です。

残り'MISSILE'数,'PROBE'数に応じたボーナス点が加算され、獲得点数が表示されます。

4. 開発環境など

ソースは、awk 版として新規に作成したものです。ThinkPad の WinXP 上のテキストエディタとコマンドプロンプトな環境でやってます。

当初の題材はビリヤードとか反射衛星砲(...ってw)なんかをイメージしていたのですが、"貧相な画面のゲーム"としてうまくまとめることができず。気が付いたらすっかり別のゲームに変わっていましたとさ^^;

近頃はOSとしてタッチパネルなUIが採用された環境もあるみたいですし、いっその事そんなUIのゲームにしてみるのが面白いかもね。

また'元ネタ'を調査するにあたって、有力な手掛りwebページ情報(※)を Zazel さんより提供いただけました事を重ねて御礼申し上げまする。
(^.^)/ どもども

※そのテラ懐かしすwebページ
→ http://www.geocities.jp/upd780c1/n80/list1/ascii/index_1984.html
1979～1987。月刊ASCIIの他にも色々。

5. その他

本プログラムは「BLACK BOX, 阿部兼幸作, 月刊ASCII 1984/10掲載」よりアイディア借用しました。'元ネタ'ゲームの作者 阿部兼幸様にはこの場をお借りしてアイディア借用の許可をお願いしたいと思います。m(_ _)m 何か問題がありましたらご連絡願います。

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。(できましたら素晴らしい改造を加えた後にTSNetへ投稿してくださいませ)

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。

以下に、スクリプト・コードを掲載します。

[MagneticRepulsion.awk]

```
##  MagneticRepulsion   written by Ｙさ

BEGIN{
  # ゲーム盤サイズ決定
  MAXSIZE=10;
  printf("¥nSize (4-%d) ?>", MAXSIZE);
  do{ max=""; getline max; }while(max<4 && MAXSIZE<max);
```

```
  # ゲーム準備
  initialize();

  # ゲームメイン
  do{ disp(0); which(); }while(bomb>0 && missile>0);
  exit;
}

END{
  # ゲーム終了
  disp(1);
  if(sc>0) sc += missile*5 + probe; # ボーナス加算
  if(bomb==0){ s1="MISSION COMPLETE"; s2="CONGRATULATIONS !!"; }
  else{        s1="GAME OVER";        s2="...YOU LOSE"; }
  printf("¥n¥n****** %s ******¥n¥n", s1);
  printf("     %s  (SCORE:%05d0)¥n", s2, sc);
}


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

##
## 初期化
##
function initialize()
{
  # 探索進行方向差分 定義 : 方向0～3=上右下左,  4=右上,5=右下,6=左下,7=左上
  dx[0]=0;  dx[1]=1; dx[2]=0; dx[3]=-1;  dx[4]=1;  dx[5]=1; dx[6]=-1; dx[7]=-1;
  dy[0]=-1; dy[1]=0; dy[2]=1; dy[3]=0;   dy[4]=-1; dy[5]=1; dy[6]=1;  dy[7]=-1;
  # 0～3方向の左右 定義
  left[0]=7;  left[1]=4;  left[2]=5;  left[3]=6;
  right[0]=4; right[1]=5; right[2]=6; right[3]=7;

  # 探索開始位置等 定義
  #      1,2,3,4,5,6,7,8,9,0
  split("A,B,C,D,E,F,G,H,I,J", NtoA, ",");
  m[0]="  ";  for(i=1; i<=max*4; ++i) m[i]="";
  for(i=1; i<=max; ++i){
    n=i;       # 上 (→下)
    testX[n]=i;   testY[n]=1;   testD[n]=2; q[(m[n]="U" NtoA[i])]=i;
    n=i+max*2; # 下 (→上)
    testX[n]=i;   testY[n]=max; testD[n]=0; q[(m[n]="D" NtoA[i])]=n;
```

```
    n=i+max;   # 右 (→左)
    testX[n]=max; testY[n]=i;   testD[n]=3; q[(m[n]="R" (i%10))]=n;
    n=i+max*3; # 左 (→右)
    testX[n]=1;   testY[n]=i;   testD[n]=1; q[(m[n]="L" (i%10))]=n;
  }
  # pos[0-2]; # 探索結果取得用変数

  # 爆弾配置など
  for(x=0; x<=max+1; ++x)
    for(y=0; y<=max+1; ++y)
      b[x,y]=0;
  srand();
  bomb=int(max/2+0.5)+rnd(int(max/2+0.5));
  for(n=1; n<=bomb; ++n){
    do{ x=2+rnd(max-2); y=2+rnd(max-2); }while(b[x,y]!=0);
    b[x,y]=-1;
  }
  missile=int(bomb*3/2);
  probe=(max-2)*(max-2); if(probe>bomb*4) probe=bomb*4;
  if((max-2)*(max-2)<=missile+int(probe/5)) --missile;
  sc=0;  # 得点
}


##
## 探索結果取得
##
function search(n)
{
  pos[0]=testX[n]; pos[1]=testY[n]; pos[2]=testD[n];
  do{ try(); }while(pos[2]>=0 && 0<pos[0] && pos[0]<max+1 && 0<pos[1] &&
 pos[1]<max+1);
  b[pos[0],pos[1]]=n;
}
function try(  f,r,l, stat)
{
  f=getKind(pos[0], pos[1], pos[2]);
  if(f==-1){
    pos[2]+=2; if(pos[2]>=4) pos[2]-=4;
    if(getKind(pos[0], pos[1], pos[2])==-1) pos[2]=-1; # 無限ループは打ち切り
    return;
  }
```

```
  r=getKind(pos[0], pos[1], right[pos[2]]);
  l=getKind(pos[0], pos[1], left[pos[2]]);
  stat=0;
  if(r==-1 && l!=-1){ --pos[2]; if(pos[2]<0)  pos[2]+=4;  stat=1; }
  if(r!=-1 && l==-1){ ++pos[2]; if(pos[2]>=4) pos[2]-=4;  stat=2; }
  if(stat!=0){
    if(getKind(pos[0], pos[1], pos[2])==-1) return;
    r=getKind(pos[0], pos[1], right[pos[2]]);
    l=getKind(pos[0], pos[1], left[pos[2]]);
    if(r==-1 && l!=-1 && stat==2 || r!=-1 && l==-1 && stat==1) pos[2]=-1;
    return;
  }

  pos[0]+=dx[pos[2]];  pos[1]+=dy[pos[2]];
}
function getKind(x,y,d,  t)
{
  t=b[(x+dx[d]), (y+dy[d])];  return ((t<0)?(-1):(t));
}

##
## 場所指定文字列変換
##
function toPos(str,  r,c,s)
{
  r=substr(str,2,1) +0;
  if(1<=r && r<=max){
    s=toupper(substr(str,1,1));
    for(c=1; c<=max; ++c) if(s==NtoA[c]) return r*100+c;
  }
  return -1;
}

##
## 場所指定
##
function which(  p,tmp,x,y)
{
  do{
    printf("¥nCOMMAND?>"); do{ tmp=""; getline tmp; }while(tmp=="");
```

```
    ## probe ?
    if(probe>0 && (toupper(tmp) in q)){
      if(--probe<0) probe=0;
      search(q[toupper(tmp)]);
      return;
    }

    ## missile ?
    p=toPos(tmp);
    # (念のため)1,2文字目を入替
    if(p<0) p=toPos(substr(tmp,2,1) substr(tmp,1,1));
  }while(p<0);
  if(--missile<0) missile=0;
  x=p%100; y=int(p/100);
  if(b[x,y]==-1){
    --bomb; sc+=100;  print "     <<< SUCCESSFUL ATTACK ! >>>¥n";
    b[x,y]=-2;
  }else if(b[x,y]!=-2)
    b[x,y]=99;
  if(bomb>missile && probe>=5){ probe-=5; ++missile; }  # 補充
}


##
## ゲーム盤表示
##
function disp(sw,  y)
{
  print "";
  y=0;     line01(y); line02(); line03(0);
  for(y=1; y<=max; ++y){
    line04(sw, y);
    if(y<max) line05(y);
  }
  y=max+1; line03(2); line02(); line01(y);
  printf("¥n### BOMB:%02d ###   ENERGY:%03d0 [PROBE:%02d / MISSILE:%02d]¥n",
         bomb, missile*5+probe, probe, missile);
}
function line01(y,  x)
{
  printf("       ");
  for(x=1; x<=max; ++x){ printf(" %s ", m[b[x,y]]);  if(x<max) printf(" "); }
```

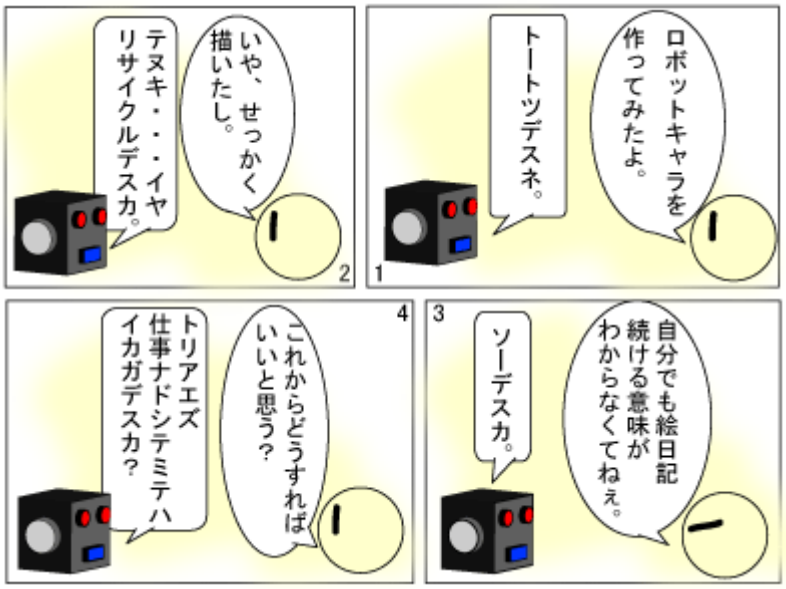
```
  printf("       ¥n");
}
function line02(  x)
{
  printf("      |");
  for(x=1; x<=max; ++x){ printf("    ");             if(x<max) printf("|"); }
  printf("|      ¥n");
}
function line03(n,  x)
{
  printf("  ----+");
  for(x=1; x<=max; ++x){ printf("<%s>", m[x+n*max]); if(x<max) printf("+"); }
  printf("+----  ¥n");
}
function line04(sw,y,  x,s)
{
  printf("%s <%s>", m[b[0,y]], m[y+max*3]);
  for(x=1; x<=max; ++x){
    s="  ";
    if(sw==1 && b[x,y]==-1) s="@@";
    else if(b[x,y]==-2) s="><";
    else if(b[x,y]==99) s="??";
    else if(b[x,y]>0) s=m[b[x,y]];
    printf(" %s ", s);
    if(x<max)
      printf("%s", (2<=y && y<max && (x==1 || x==max-1))?("#"):("|"));
  }
  printf("<%s> %s¥n", m[y+max], m[b[max+1,y]]);
}
function line05(y,  x)
{
  printf("  ----");
  for(x=1; x<=max; ++x)
    printf("%s", (2<=x && x<max && (y==1 || y==max-1))?("+===="):("+----"));
  printf("+----  ¥n");
}
```

（投稿日付: 2009年11月27日）

よしおさんとロボ太 ～ 原点の６編 ～

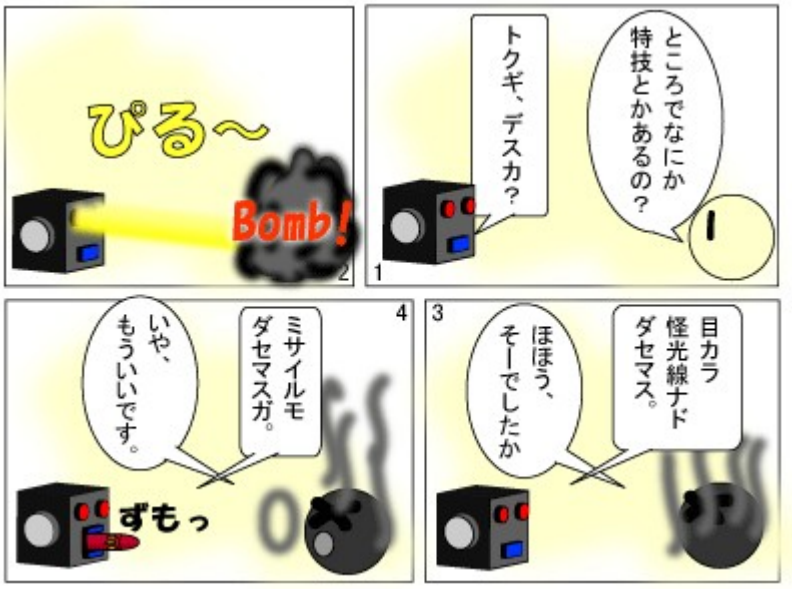
海鳥作

「作ってみた」



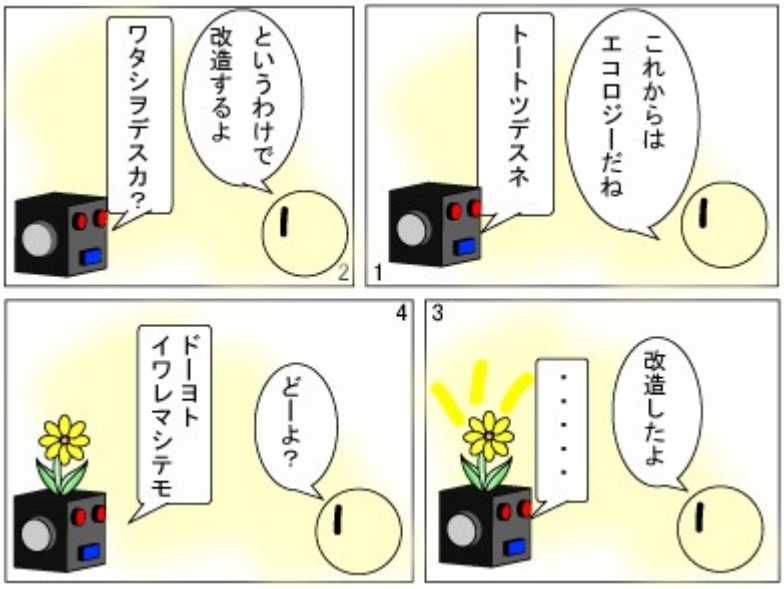
手抜きキャラ出現。

「特技」



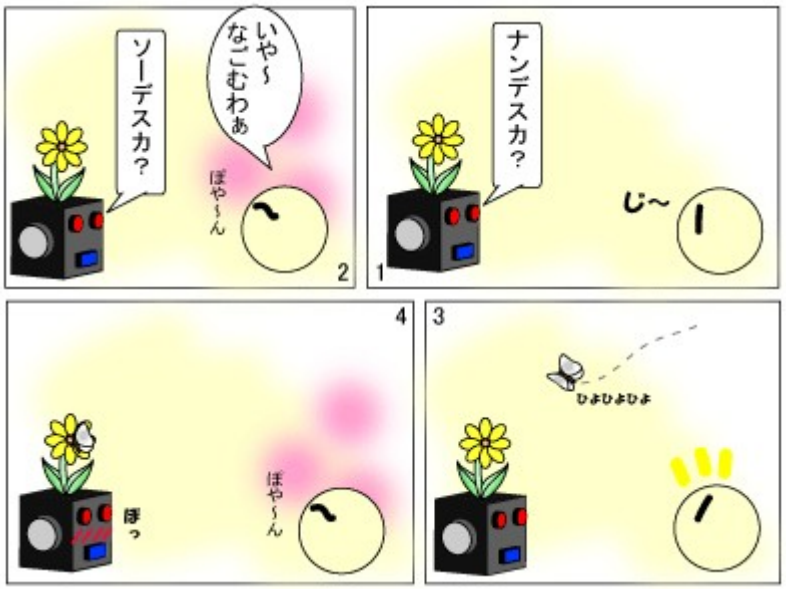
ミサイルの寸法がおかしい。

「エコロとココロ(前編)」



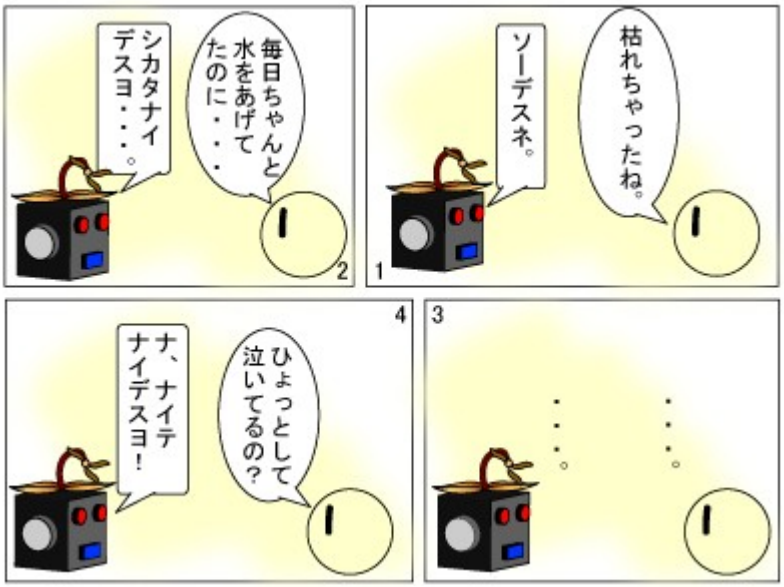
地球にやさしい(たぶん)

「エコロとココロ(中編)」



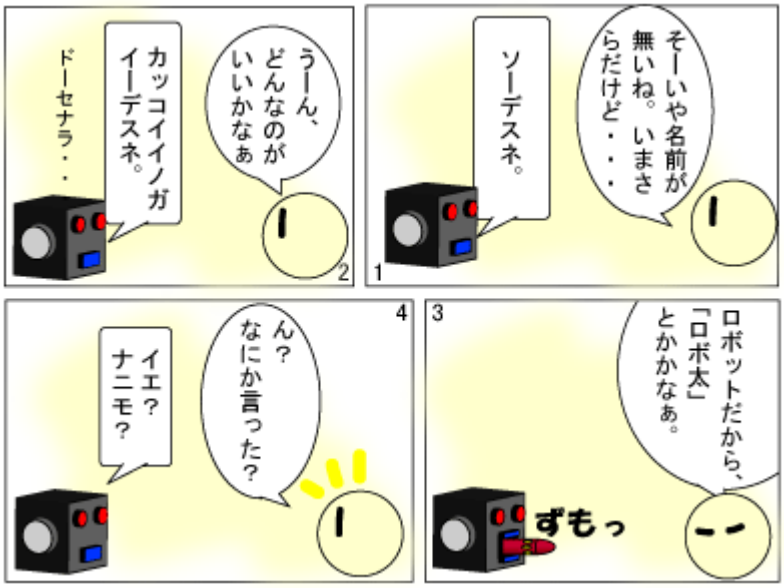
ツンデレロボ(違)

「エコロとココロ(後編)」



エコロとココロ。

「命名」



お気に召しませんでしたか？

(転載日付: 2009 年 11 月 29 日)

眠り姫

きしだあつし

むかし、むかし。あるところに。

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

わたしには生まれつき手足がなかった。

もちろん当時のわたしにはそんな概念すら持ち得なかったのだから、その意味するところなど知ることもない。

母はそんなわたしを特に悲しがるわけでもなく、かといっていたわる訳でもなくただたんたんとその現実を見守りわたしに告げたものだった。

（あなたは大切な使命をもって生まれてきたのよ）

と。

当時のわたしはまだその使命についてなにも知らなかったし、いまもそれは変わりない。ただ、未来に托されたプロジェクトがわたしの中にあるということだけは知っていた。それはいまのわたしには知るよしもないことであり、いずれ時が訪れたときにわたしはその内容について知ることになるはずだ。

母はその時が来るまではゆっくりと眠りなさい、といつもわたしに云った。寝る子は育つっていうのよ、とも云った。けれどもわたしにはなぜ寝ている間に成長できるのか分からなかった。もちろん今もその答えはでていない。

わたしが生まれたのは十月二十三日一時三十六分四十三秒。母が記念だからと云って記録を残してくれたので、わたしはいつでもこの日を思い出すことが出来る。深夜だったにも関わらず母は興奮のあまり寝つくことも出来ずにいたという。そしてしばらく考えた末にわたしにひとつの名前をくれた。ローザというのがその名前だ。簡潔でやさしくて、けれど情熱も秘めている、そんな名前だという。

母はとてもわたしを気に入ってくれた。そして彼女が次に行なったのは、わたしのクローンを作ることだった。クローンを作ることは決して難しいことではない。むしろごく簡単に行なえることだ。わずかの操作でわたしと寸分違わぬクローンを作り上げることが

出来る。

次に母は、まだ幼いわたしたちをあちらこちらに里子に出した。もちろん、わたしもその中のひとりに過ぎない。

わたしたちは、たどり着いた新たな地で適当な宿主を見つけては新たな卵を産み付けていく。そしてまたあらたな地を目指して旅立つ。時としてあらたな地がすぐには見つからないときもある。そんなときはひたすらじっと眠って次の機会を待つ。

そうしてわたしたちはとてつもなく増殖を繰り返す。

卵を産み付ける相手は小さすぎないことが大切だ。より大きな相手であれば簡単には見つからない。もちろん、だからといって単純に産み付けるわけではない。その事実を悟られないように多少の工作をすることもある。一見したところではわたしたちの存在に気づかれないような工作だ。

とはいえ、用心深い場所においては、疑わしいといっては駆除されてしまう場合もあって、わたしたちの存続は決して易いことばかりではない。

だからこそ、わたしたちはより繁殖しやすい場所を求め、より生存確率を高めるために新天地を常に求め続けていた。

時々見知らぬわたしたちの仲間に出会うこともあった。卵を産みつけようとするとその仲間がすでに産みつけたあとで、わたしの行為を強固に拒絶することもあった。仲間とはいっても同じ種族であるであろうというだけのことで、はじめて出会うものたちだ。詳細に調べて比べるわけではないが、わたしと似たものやまったく異なる種族もいる。

ごく稀にはわたしの亜種に出会うことすらあった。長い旅のさなか、どこかで交配を繰り返した結果なのだろうか、わたしと同じ匂いはするものの、細かな部分が異なっていた。

そうした似た種族の末路を確認したり、みずから危うく殺されかけたりを繰り返しながら、今ようやく安定した環境に身をおくことができたようだった。この環境であればその日まではゆっくりと眠って過ごすこともできそうだ。

わたしはしばしの眠りにつくことにした。その日まで。

いよいよその日がやってきた。まさに今日、この日だ。まもなく母から教えられた時刻がやってくる。

まだわたしには何をなすべきなのかはわからないが、その時刻がくればそれはおのずとわかるはずだ。何も迷うことも悩むこともない。母があらかじめ指示してくれたことを実行するだけだ。そのすべてはわたし自身の中にあるはずだ。

時間だ。

瞬時に母の指示がわたしの中をかけめぐる。なすべきことはさほど多くはない。きわめて単純なことだ。そう、すべてはあなたのためよ。母のことじゃないわ。あなたよ、あなた。

こんな時間まであなたは何をしているというの？　さっさと家に帰るべきよ。そうよ。今のわたしにはわかっている。自分が何をするのか。手伝ってあげるわ。さあ、これでどうかしら。

『メリークリスマス。こんな時間なのに、まだ仕事をしているの？　さっさとお帰りなさい』

＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

少しひんやりしてきたオフィスのなかで、男は、ディスプレイに表示されたメッセージの下に、次々と流れていく削除ファイルのリストを、ただ呆然と見つめていた。

むかし、むかし。あるところに。

（投稿日付：2009年11月30日）

スクリプティング言語の周辺

jscripter

本稿では、スクリプティング言語あるいはコンピューティングの周辺について現在までの状況を振り返り、近い将来を展望してみたいと思う。

夢のようなプログラミングが現実に

僕等は IBM のメインフレームから DEC などのミニコン、ワークステーションへ、そしてついに PC へと時代が変化するのをわくわくしながら眺めていた。コンピュータが自分の手元で動かせる時代が到来するのを待ちに待っていた。パーソナルコンピュータの時代だ。ジョブスやビル・ゲイツの活躍を期待を持って眺めていたのだ。その時が来た。

21 世紀に入って来年は 10 年目である。一般人がパソコンを所有し、アスキー誌などで AWK などのスクリプティング言語を知るようになったのは 1990 年頃のことである。当時、AWK や Perl が日本語化され、jgawk や jperl が生み出されていた。僕はテキストをデータとして取り扱えることに単純に感動していた。自分のコンピュータでテキストデータを自在に加工してデータベースを作れると。それから 20 年近くが経過したことになる。

その間に様々なことが起った。Steve Jobs は Apple Computer から飛び出し、1985 年に NeXT Computer を生み出していた。Tim Berners-Lee は 1990 年に NeXT 上で World Wide Web を発明した。これは時代を飛躍的に変化させる発明だった。そこで、CGI が生まれた。HTML とテキスト処理が結びついたのである。WWW によってインターネットが普及し、パソコン通信は衰退した。同時にマイクロソフトの MS-DOS は Windows へと進化し、Windows95 は Web ブラウザが普及する基盤となった。MS-DOS で動いていた jperl などのスクリプティング言語も次第に Windows で動くようになっていった。AWK や Perl 以外のスクリプティング言語、Python や Tcl/Tk も日本語で動作するようになり、日本生まれの Ruby が登場した。現在では、求めれば多様な言語を誰もが利用できるようになっている。20 年前は C 言語か Basic 言語ぐらいが一般的な選択肢であったことを思うと隔世の感がある。

オライリー・ジャパンの Clinton Wong 著「Web クライアントプログラミング with Perl」が出たのは 1997 年(原著も 1997 年)。著作のきっかけは 1995 年に既にあったそうだから、この種の解説本としてはもっとも早いものではないかと思われる。僕が購入したのは、おそらく世紀末に近くなってからだろう。同じくオライリーの「CGI プログラミング」の初版が出たのが 1996 年だが、僕が持っているのは、Scott Guelich、Shishir Gundavaram、Gunther Birznieks 著「CGI プログラミング 第 2 版」(2001 年;原著: 2000 年)だ。そうして、僕は Web クライアントプログラミングから CGI プログラミングに入っていった。テキスト処理の対象は、パソコン通信のログから、HTML/XML/RDF に変化した。1990 年代半ば以降、ネットワークを経由する自在なテキスト処理が現実のものとなったのである。

メディアとしてのインターネット

21 世紀に入ると、Google などの検索エンジンによってインターネットの利用は強力に推進された。僕は 2001 年 5 月 1 日の日記に「メディアとしてのインターネット」と題して次のように書いている。

『インターネットのメディアとしての可能性は大きい。今や膨大なホームページからなるデジタル情報の大海から、Google に代表される強力な検索エンジンによって、貴重な情報をすくい取ることができる。

本とコンピュータ誌によれば書籍の売上は以前と比べると随分落ちているそう で、みんな本を読まなくなった。これはメディアが多様化したためだ。テレビや最近ではテレビゲームの出現の影響が最も大きかっただろうが、人間の一日の時 間は長くなっていない。ここでさらにインターネットが出てきて、あらゆるメディアがここに集約される可能性が出てきた。テレビ、ビデオ、ラジオ、音楽、ゲーム、映画、本、電話、FAX、メール、チャット、掲示板、ニュース、新聞、美術館、博物館、ギャラリー、フォーラム、百科事典、辞書、翻訳サービス、教育、・・・

放送と通信の融合ということが言われて久しい。当時はといっても数年前だが地上波デジタル放送が主役になるだろうと思われていた。いまやブロードバンドどころかナローバンドでも動画の配信の可能性が出てきている。Palm でさえ、gMovie で動画が受信できる時代になりつつある。人間はよりいっそう-ながら族-にならざるを得ない。携帯を耳に押し当てて歩くのは既にありふれた現実であり、これからは Palm でニュースを見ながら会社に通うのだろうか。

デジタルの波を断ち切って、ゆっくりと本を読む時間を持つのが一番の贅沢かもしれないが、インターネットに接続しているのがどうしても長くなるこの頃である。読む本といえばコンピュータ関連ばかり(^^;)常時接続が普及すれば社会の様相がかなり変わる可能性があるだろう。』

常時接続が普及した 2003 年ぐらいから、世界のあらゆる情報がインターネットの Web サイト上に存在するようになった。様々な世界的な会議や展示会における発表、製品発表などだけでなく、大学の授業内容さえ、公開された。画期的なことである。大学の授業の内容は、OpenCourseWare と呼ばれ、MIT で始まった。日本の大学も追従している。米国では政府の活動やそれによって得られた情報も開示されている。そのような情報を活用するアプリケーションのコンテストさえ開かれるという具合である。好むと好まざるとに関わらずインターネット上に情報がなければ、実世界に存在していないと見なされかねないという状況に至っている。

Steve Jobs は NeXT を Apple Computer に売却し、自ら復帰した。買収したピクサー・アニメーション・スタジオも最終的にディズニーに売却した。iPod という小さな音楽デバイスが iTunes という優れたソフトウェアと共に Apple Computer を立て直した。そして、

ケータイに参入。iPhone を成功させた。Mac も売れているが、社名は Apple に変った。コンピュータという概念で Apple のビジネスを括るのは最早相応しくない。アップルはハードウェアとソフトウェアとインターネットをブレンドして新しいメディア産業を生み出した。

Google Wave

今や、Apple と並んで、インターネットで圧倒的な存在感を示しているのが Google である。検索エンジンから出発して、Web アプリとしては Google Map、YouTube など。クラウドとしては、Gmail、iGoogle、ディスカッションや情報の共有のための Google Group や Google Code がある。デスクトップには、Google Desktop、Picasa、Google Chrome という Web ブラウザなど。さらに Chromium OS、ケータイ用の Android とハードウェアにも接近している。

知的生産の観点から今後注目されるものに、Google Wave がある。ビデオや写真など多様なメディアをミックスできる一種の SNS であるが、Web 上の生産的なコラボレーションツールとして発展する可能性があるだろう。

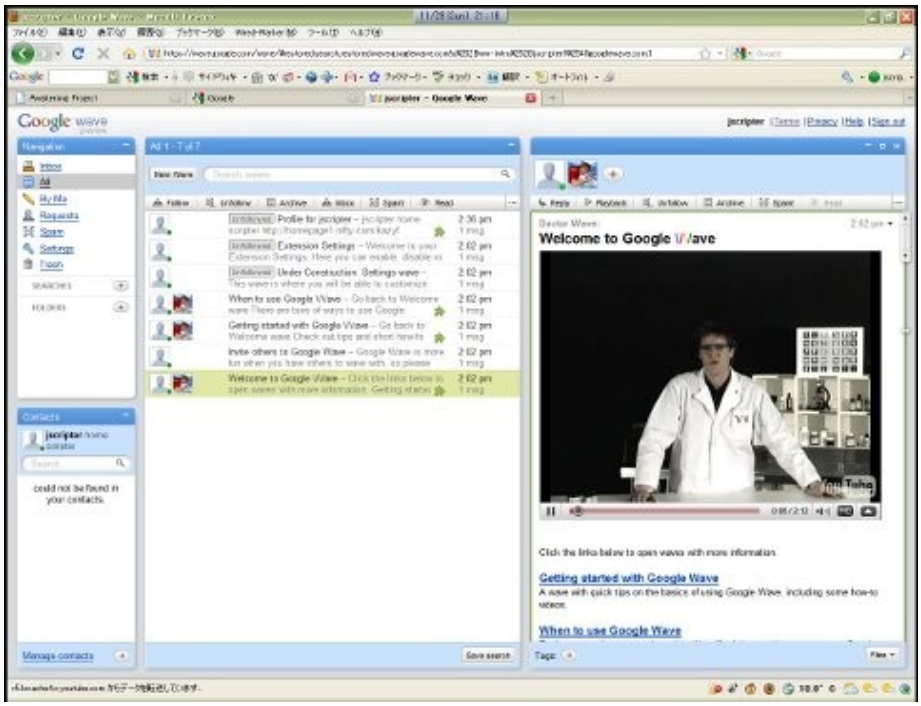


図. Google Wave Preview

Rakudo Perl 6

さて、21 世紀のスクリプティング言語はどうなったか。それほど大きな変化があったかというと今のところそれほどでもないと思う。実際できることはコンピュータでできることができるだけだから当然ではある。今後 20 年のプログラミングの基盤となるべく、

21 世紀初頭に構想された Perl 6/Parrot はまだ未完成のままである。ライブラリの充実などを含め、実用に供するためには、後、数年を要するだろう。しかし、既に、20 年から 100 年と言いなおされているから問題はない。今、Perl を実用で使うなら、Perl 6 の仕様の影響を受けて変貌しつつある Perl 5.8.9 や Perl 5.10.1 を使えば良いのだから。

Perl 6 の実装として、Rakudo Perl 6 が登場しつつある。11 月 20 日に perlpilot 氏が Perl 6 本を Rakudo.org に投稿している。40 ページのチュートリアルのような本だが、Rakudo を試してみるには手頃だろう。僕はようやく欲しかったものが手に入ったと喜んだ。下記の URL から入手しよう。

More Perl 6 Book | Rakudo.org
http://rakudo.org/node/60

最初のスクリプトサンプルは、テーブルテニスのトーナメントの結果から、誰が何試合勝って、何セット取ったかを勝利数順、獲得セット数の順に降順にソートするものである(http://cloud.github.com/downloads/perl6/book/book-2009-11.pdf: Chapter 2 The Basics, 4-8ページ)。正確には昇順にソートしてリバースでひっくり返している。次のような試合結果を記述したテキストファイルが入力となる。

[scores.txt]

```
Beth Ana Charlie Dave
Ana vs Dave | 3:0
Charlie vs Beth | 3:1
Ana vs Beth | 2:3
Dave vs Charlie | 3:0
Ana vs Charlie | 3:1
Beth vs Dave | 0:3
```

次のスクリプトはPerl 5で動かそうとするとエラーになる。最初にある「use v6;」はRakudoのスクリプトであるという目印なのである。ただ、Rakudoで動かす場合にはなくてもエラーにはならない。それ以外の部分は、PerlやRubyを知っていれば、何をやっているか、イメージは浮かぶだろう。Ruby風の「.」で区切ってメソッドをオブジェクトに後置していく書法が新しい。これは、dotty method operatorと呼ぶらしい。Perl 5のユーザーであれば、ハッシュ(連想配列)の各要素の書き方が違うことに戸惑うかもしれない。Perl 5ならハッシュの要素であっても、$sets{$p1}のように先頭は「$」で始める。

[scores.p6]

```
use v6;
my $file = open 'scores.txt';
my @names = $file.get.split(' ');
my %games;
my %sets;
for $file.lines -> $line {
      my ($pairing, $result) = $line.split(' | ');
      my ($p1, $p2) = $pairing.split(' vs ');
      my ($r1, $r2) = $result.split(':');
      %sets{$p1} += $r1;
      %sets{$p2} += $r2;
      if $r1 > $r2 {
            %games{$p1}++;
      } else {
            %games{$p2}++;
      }
}
my @sorted = @names.sort({ %sets{$_} }).sort({ %games{$_} }).reverse;
for @sorted -> $n {
      say "$n has won { %games{$n} } games and { %sets{$n} } sets";
}
```

実行すると次のような結果が得られる。

```
C:¥Scripts¥perl6>perl6 scores.p6 scores.txt
Ana has won 2 games and 8 sets
Dave has won 2 games and 6 sets
Charlie has won 1 games and 4 sets
Beth has won 1 games and 4 sets
```

ここまで来ると、Perl 5で書いて比較してみたいと思うだろう。試してみた。引っ掛かったのは、split関数の分割するパターンの記述を'|'ではなく、'¥|'とする必要があるところだ。Perl 5の場合、splitの最初の引数に文字列を与えた場合には正規表現として解釈されるからだ。Rakudoでは単なる文字列と解釈されるためか、エラーになる。ダブル・クォート内で{...}で括る変数展開のさせ方は、ハッシュの要素の書き方を変えたために必要となっているようだ。

[scores.pl]

```
open(IN, "scores.txt");
chop(my $names = <IN>);
my @names = split(' ', $names);
my %games;
my %sets;
while(<IN>){
      chomp;
      my ($pairing, $result) = split(' ¥| ', $_);
      my ($p1, $p2) = split(' vs ', $pairing);
      my ($r1, $r2) = split(':', $result);
      $sets{$p1} += $r1;
      $sets{$p2} += $r2;
      if($r1 > $r2){
            $games{$p1}++;
      } else {
            $games{$p2}++;
      }
}
close(IN);
my @sorted = sort {
       $games{$b} <=> $games{$a} or $sets{$b} <=> $sets{$a}
} @names;
for (@sorted){
      print "$_ has won $games{$_} games and $sets{$_} sets¥n";
}
```

出力結果は次のようになる。ソートの部分は勝利数順、獲得セット数について降順にソートしている。その結果、出力が少し異なる。Beth と Charlie の順序が逆になる。昇順にソートして、リバースすれば、当然のことながら同じ結果が得られる。リバースは配列を逆順に並べ替えるだけで、ソートするわけではない。Rakudo の場合、単純に降順にソートしたい場合にはどのように書くのだろう。

```
C:¥Scripts¥perl6>perl scores.pl scores.txt
Ana has won 2 games and 8 sets
Dave has won 2 games and 6 sets
Beth has won 1 games and 4 sets
Charlie has won 1 games and 4 sets
```

このレベルの記述の違いはそれほど本質的なものではない。Rakudoは、2010年の春に「Rakudo *」あるいは「Rakudo Star」として正式にリリースされる。また、次の機会に

Rakudoのオブジェクト指向の詳細などについて取り上げよう。

Rakudo/Parrotのインストール

話は前後するが、Rakudo/Parrotのインストールについて簡単に言及しておこう。既に配布システムは整備されてきている。

Browse parrot-win32 Files on SourceForge.net
http://sourceforge.net/projects/parrotwin32/files/

から、setup-parrot-1.8.0.exeをまずインストールし、setup-parrot-1.8.0-rakudo-23.exeを同じフォルダに上書きインストールすればよい。環境変数のPathに、インストールフォルダのbinが付け加えられているはずだ。通常、「C:¥Parrot-1.8.0¥bin」になるだろう。適当なフォルダのコマンドプロンプトで実行してみよう。次のようになる。

```
C:¥Scripts¥perl6>perl6 -v
This is Rakudo Perl 6.

Copyright 2006-2009, The Perl Foundation.
```

スクリプティング言語の将来

Perl 6/ParrotがPerlの話題の中心になって久しいが、最近は、www.perl.orgはホームページを改装して、Perl 5を全面に打ち出している。Current Perl versionは、5.10.1とアナウンスされている。実際上、Rakudo Perl 6は未完成なので当然である。Perlユーザーは、Perl 5を使い続けている。

The Perl Programming Language - www.perl.org
http://www.perl.org/

スクリプティング言語のユーザーは、もちろん他のプログラミング言語のユーザーも同じだろうが、新しいバージョンや新しい言語にユーザーは何を期待するのか。二つある。実行スピードとプログラミングの容易さである。しかしながら、ユーザーはバージョンアップや新しい言語の登場には期待もしているが、日常的に問題がない場合、あるいは特に問題意識がない場合には、アップデートしたり、新しい言語を学ぶ必要性がどこまであるのかという問題に直面するものである。また、PCのハードウェア性能の向上によって、スクリプティング言語の処理能力は20年前と比較すると劇的に向上しており、実用的にはかなり満足するものとなってきている。したがって、おそらく、パーソナルユースにおいては個人の好みや経験・習熟度、用途に適したライブラリの存在の有無などが使用言語を決定する主要な要因となっていくだろうと思われる。

PCの最もハードな用途はアクセスの集中するサーバーと考えられる。サーバー資源を有効に利用するためには、スクリプティング言語を含むプログラミング言語のパフォーマンスが重要である。最近ではTwitterのシステムがRubyで構築されていることが話題になった。僕が注目してみているのは、Googleの動向である。プログラミング言語として何を採用するか。今のところ、JavaとPythonである。最近では、特にパフォーマンスを必要とする部分(Native Clientなど)には、GoというPlan 9から派生したプログラミング言語を提案しつつある。これはクライアントのWebブラウザ側のパフォーマンスを向上させることによって、サーバー-クライアント・システムのトータルのパフォーマンスを確保する手段とも考えられる。一時、Erlangも注目されたが、Goによる並列処理プログラミングが今後の注目点になるのかもしれない。

Go: A New Concurrent Systems Programming Language from Google
http://www.ddj.com/go-parallel/blog/archives/2009/11/go_a_new_concur.html
The Go Programming Language
http://golang.org/

P.N. ジョンソン・レアード著『メンタルモデル―言語・推論・意識の認知科学』(産業図書、1988年)の第16章「意識と計算」の最後のほうには次のように書かれている(561ページ)。

『・・・もし機能主義の議論が正しいとすれば、意識は、ある種の並列アルゴリズムが有する性質である。そして、コンピュータも意識を持つことができる。ただし、いまだかって、コンピュータが意識を持つことについてその直接的な証拠がみつけられたことはない。』

おわりに - 知識表現・思考のツールとしてのスクリプティング

僕はぼんやりとだが、知識表現・思考のツールとしてのスクリプティングということを考えている。それはオブジェクト指向やフレームワークという概念と結びつくはずだと思っているのだが、まだ具体的な描像を結ばない。まあ、それほど焦らずとも日々は過ぎていく。いつの日か、あっ、これかなと思う日が来るのを楽しみにしている。

(投稿日付: 2009年11月30日;改訂: 2009年12月5日)

Ruby/Tk してみませんか

きしだあつし

Ruby をはじめとしたスクリプト言語は手軽に書いて手軽に実行できるという便利さがうれしい反面、基本的にはコマンドラインで操作することになり、プログラムによっては視覚的に使える GUI 環境があると便利なのにと思うこともあります。

そんなときに便利なのが Tcl/Tk を使うことです。GUI ツールキットである Tk を Ruby で使うためには Ruby/Tk が必要ですが、Ruby に同梱されているのですぐに使うことができます。Tcl/Tk は別にインストールする必要があります。Windows 環境であれば、Active Tcl をインストールするのが一番簡単でしょう。

http://www.activestate.com/activetcl/

ActiveState から Windows 用のインストーラーパッケージをダウンロードして実行します。メッセージにしたがってすすめていけばインストールは完了します。

Windows で使用する Ruby では、ActiveScriptRuby がお薦めです。最新のバージョンにいち早く対応してくれていることや、余分なものが入りすぎていないために比較的コンパクトであることもあり、現状では唯一の選択といってもよいくらいです。

http://arton.hp.infoseek.co.jp/indexj.html

基本的にはこれで Ruby/Tk が使えるはずですが、実行しようとしてエラーがでることがあるかもしれません。なにかのファイルが見つからないなどがあれば適宜インストールします（エラーメッセージなどから検索して必要なファイルを探してみてください）。

Hello World!

まずはお約束の Hello World! を表示してみましょう。

#sample01

```
require 'tk'
TkLabel.new('text' => 'Hello World!').pack
Tk.mainloop
```

画像:1　sample01

　標準ではサイズを指定してないので、表示するテキストに応じた大きさのラベルが、同じくそれに沿った大きさのウィンドウに表示されています。

　ラベルの大きさを指定してみます。文字数で指定します。

#sample02

```
require 'tk'
TkLabel.new('text' => 'Hello World!', 'height' => 5, 'width' => 20).pack
Tk.mainloop
```

画像:2 sample02

　ラベルの大きさが変わり、それにともなってウィンドウの大きさも変わったでしょうか。

　ボタンを追加してみます。

#sample03

```
require 'tk'
t = TkVariable.new
TkLabel.new('textvariable' => t, 'width' => 10).pack
TkButton.new('text' => 'Click Me!',
             'command' => proc{ t.value = 'Hello World!' }).pack
Tk.mainloop
```

画像:3　sample03

ボタンを押すとラベルにメッセージが表示されたでしょうか。

今と同じものを、今度は横並びに表示してみましょう。

#sample04

```
require 'tk'
t = TkVariable.new
TkLabel.new('textvariable' => t, 'width' => 10).pack('side' => 'left')
TkButton.new('text' => 'Click Me!',
  'command' => proc{ t.value = 'Hello World!' }).pack('side' => 'left')
Tk.mainloop
```

画像:4　sample04

pack はジオメトリマネージャと呼ばれるもので、ラベルやボタンなどさまざまな部品（ウィジェット）の配置を管理しています。指定できるのは top bottom left right の４つの位置です。標準では top として処理されます。

この配置についてはやや独特で具体的には実際にあれこれ配置してみて実感するのが一番ですが、基本的にまだなにも配置されてない領域のなかの位置関係を示します。はじめに top に配置すると、基本的には次に配置するウィジェットがその同じならびに隣り合うことはありません。top で配置されたウィジェットはその領域内の上の場所（左右のはじまですべて）を確保してしまいます。

今の例で配置を変えて試してみましょう。

#sample05

```
require 'tk'
t = TkVariable.new
TkLabel.new('textvariable' => t, 'width' => 10).pack('side' => 'top')
TkButton.new('text' => 'Click Me!',
  'command' => proc{ t.value = 'Hello World!' }).pack('side' => 'right')
Tk.mainloop
```

画像:5 sample05

ボタンはラベルの隣に配置されません。ラベルで上側全体（左右いっぱい）に確保された残りの領域（すなわち下半分：厳密に半分ではありませんが）において右位置を確保するだけです。

左右に配置するには sample04 のようにする必要がありますが、では、次のようにしたら「左上にラベル、右上にボタン、下側にもうひとつボタン」が配置されるでしょうか。

#sample06

```
require 'tk'
t = TkVariable.new
TkLabel.new('textvariable' => t, 'width' => 10).pack('side' => 'left')
TkButton.new('text' => 'Click Me!',
  'command' => proc{ t.value = 'Hello World!' }).pack('side' => 'left')
TkButton.new('text' => 'New Button').pack('side' => 'bottom')
Tk.mainloop
```

画像:6 sample06

はじめに left を確保した段階で表示領域の左側の上から下まで確保されているため、

仮にウィンドウの大きさがもっと大きくてもその確保された領域は変わらないので上下に配置されることはありません。あくまでもまだ確保されていない領域のなかの配置位置を確保します。

このような pack の特徴はよく理解しておく必要があります。

フレームがポイント

pack の特徴は特徴として、ではいったいどうしたらもう少し柔軟な配置ができるでしょうか。そのために使うのが frame です。ウィジェットの配置を領域確保の特徴に応じて分割し、そのまとまりごとにフレームで確保し、そのフレームの中にそれぞれのウィジェットを配置することである程度解決できます。

sample06 を意図した配置にしてみましょう。

#sample07

```
require 'tk'
t = TkVariable.new
f1 = TkFrame.new.pack
TkLabel.new(f1, 'textvariable' => t, 'width' => 10).pack('side' => 'left')
TkButton.new(f1, 'text' => 'Click Me!',
  'command' => proc { t.value = 'Hello World!'}).pack('side' => 'left')
f2 = TkFrame.new.pack
TkButton.new(f2, 'text' => 'New Button').pack
Tk.mainloop
```

画像:7 sample07

フレームの pack も忘れずに。フレームのなかにフレームを入れ子にしたりといったことも当然できます。

Visual Basic や Delphi などで便利に使われはじめた、ボタンやラベルなどの部品をウィンドウにマウスで自由に配置してプログラムするのと同じというわけにはいきませんが、その特徴をきちんと理解すれば手軽に GUI 環境を作ることが可能になります。

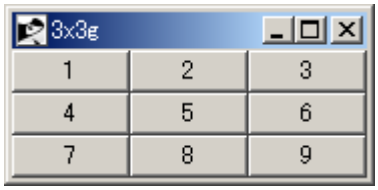
それまでコマンドラインで使っていたものの、各種のオプション指定が煩雑であるとか、または自分以外の操作に不慣れな第三者の使用を想定したときなど、やはり GUI 環境で視覚的に理解できることの効果は大きなものです。

配置の特徴を知ってフレームをうまく活用することが、Tk 使いのポイントではないかと思います。

最後に簡単なゲームを作ってみましょう。プログラムを実行すると 1 から 9 までの数字が書かれたボタンが 3 列 × 3 行に配置された画面が表示されます。ボタンをクリックするとボタンが消えていきますが、ひとつのボタンに爆弾がセットされています。最後まで爆弾を選ばずにボタンを消していけるでしょうか。

#sample08

```
require 'tk'
f = Array.new(3)
b = fb = Array.new(9)
bomb = rand(9)
3.times do |i|
  f[i] = TkFrame.new.pack
  3.times do |j|
    k = i * 3 + j
    fb[k] = TkFrame.new(f[i]).pack('side' => 'left')
    lb = TkLabel.new(fb[bomb], 'text' => 'BOMB!', 'width' => 7)
    b[k] = TkButton.new(fb[k]) {
                                text "#{k + 1}"
                                width 7
                                command proc {
                                          lb.pack('fill' => 'x') if bomb == k
                                          b[k].destroy
                                         }
                                pack
                                }
  end
end
Tk.mainloop
```



画像:8　sample08

おわりに

今回紹介したのはほんの入り口でしかありませんでしたが、様々なウィジェット（部品）が用意されており手軽に GUI を持ったプログラムを作ることができます。これまでコマンドラインで使ってきたプログラムに GUI を追加することも（比較的）簡単にできます。

ただ、以前から言われているように公式なドキュメントの整備は必ずしも十分とはいえず、数少ない書籍やネットにあがっているいくつかのサイト情報などしかないために、まだまだ情報不足といえそうです。

幸いにして Ruby/Tk の普及に尽力されている永井秀利さんがドキュメント整備のためのサイトを用意され、順次充実に向けて活動されているようです。まだまだはじまったばかりで情報は少ないですが、セミナー資料などはひととおりの基礎を知るために十分な内容ですので参考にされるとよいと思います。

http://www.dumbo.ai.kyutech.ac.jp/~nagai/cgi-bin/RubyTk/

日常のちょっとした不便や問題を解決するために、Ruby をはじめとしたスクリプト言語でさくさくとプログラムし、さらにそれを便利に使うために Ruby/Tk で GUI を追加すれば、より快適に使えるはずです。簡単便利を手助けしてくれる、それが Ruby/Tk かもしれません。

Ruby を使っているあなたも、これからというあなたも、Ruby/Tk してみませんか。

（投稿日付: 2009年12月5日）

Sjis ソフトウェアってなあに？

稲葉 準

■Sjisソフトウェアってなあに?

Sjis ソフトウェアは、日本語のテキストや日本語で名付けられたファイルを手軽に扱うための Perl プログラミング環境を提供するソフトウェアです。

従来、JPerl や Encode モジュールを使って(手間をかけて)実現していた仕事の多くは、このソフトウェアを使えば、簡潔に読みやすく記述することができます。

このドキュメントは主に Microsoft Windows ユーザを読者として想定していますが、Sjisソフトウェアは他のオペレーティングシステムでも動作する可能性があり、幅広いアーキテクチャに対して移植性のある日本語情報処理モデルの提供を目指しています。

Sjis ソフトウェアは現にかなり人気を集め、その人気はさらに高まりつつあります。その理由はもちろん機能的にシンプルだからですが、それ以外に、安価に入手できることが挙げられます --- 実際、Sjis ソフトウェアは自由に(そして無料で)入手することができます。かりにこのソフトウェアが使えるようになっていなくても、自分で入手して簡単にインストールすることができます。

(早く標準モジュールになりますように！)

初期の Sjis ソフトウェアは、ダブルクォート文字列やダブルクォート風の正規表現内のシフトJIS文字の第2オクテットをエスケープするための単なるフィルタとして設計されていました。

Sjis ソフトウェアは進化するにつれて、ファイルをテストする、ファイルの情報を取得する、ファイルを削除するなど、日本語で名付けられたファイルそのものを容易に操作できるようになりました。

もちろん JPerl や Encode モジュールを用いても、これらを実現することはできますが、そのようなプログラミングは難しく、できあがったプログラムは見苦しいものになってしまうでしょう。なぜなら、JPerl が得意とするローカライゼーション(地域化)はEncodeモジュールにとって不得手な分野であり、一方、現在更新されていない JPerl はCPANにある perl5.006 以降向けの便利なモジュールを利用できないからです。

Sjis ソフトウェアは JPerl と Encode モジュールの間のかなり大きなギャップを埋めるものです --- このソフトウェアは両者の得意分野を(単一のソフトウェアで)カバーしているので、Encode モジュール環境と JPerl プログラミングのギャップを橋渡しできるのです。

Sjis ソフトウェアは JPerl とよく似たインタフェースをしているので、すでに JPerl を知ってる人にとってはこのソフトウェアを学ぶことはたやすいはずです。また見方を変えれば、あなたの最終的な目標が Encode モジュールの使い方をマスターすることであるなら、Sjis ソフトウェアに関する知識はその際、大いに役立つでしょう。

■ さっそく使ってみよう！

以下の3ステップで利用することができます。

1. CPAN の http://search.cpan.org/dist/Sjis/ から最新の Sjis.pm と Esjis.pm を取得します。

2. Sjis.pm と Esjis.pm を C:¥Perl¥site¥lib へコピーします。

3. 作成するPerlスクリプトに "use Sjis;" と記述します。

以上でスクリプトの文字列リテラルにシフトJISを記述することができます。

便利なユーティリティとして perl58.bat と perl510.bat もあります。このバッチファイルは perl5.005 を主に使っていて、perl5.008 や perl5.010 のパスを環境変数 PATH に設定していないときにディスクからこれらの実行プログラムを探して実行します。Sjis ソフトウェアと組合わせることで、あたかも jperl5.008 や jperl5.010 のようになるわけです。

■ Sjis ソフトウェアの特徴

このソフトウェアの特徴をリストします。

<u>優れている点</u>

(1) データの上位互換性
→ 過去に蓄積されたデータをそのまま処理することができます

(2) スクリプトの上位互換性
→ 過去に作成されたスクリプトは、過去に意図したように動作します

(3) 文字列の状態を表す変数がない
→ プログラミングの負担が増えません

劣っている点

(1) 国際化されていない
→ ISO/IEC 8859-1(Latin-1) を扱うことができません

以下に詳しく見てみましょう。

(1) データの上位互換性

半角カナを扱えます！

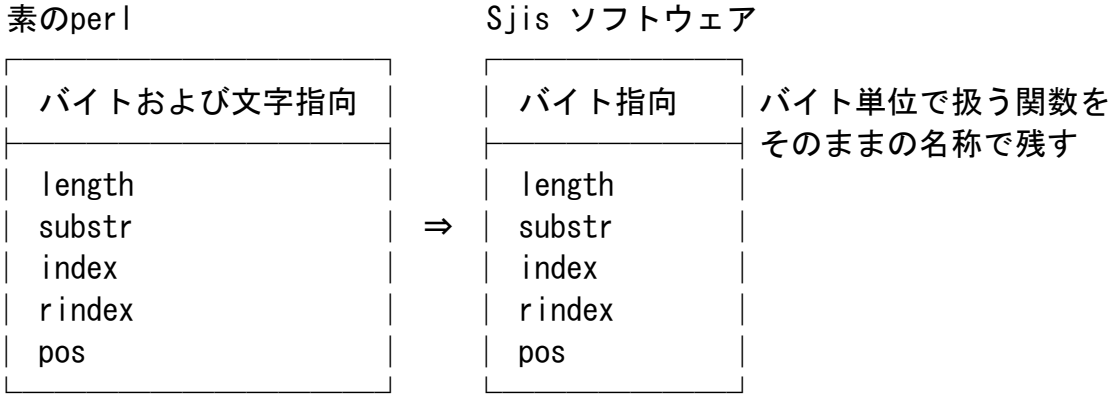
これは Sjis ソフトウェアの特徴というよりは、処理対象としているシフトJISの特徴です。シフトJISの最大の利点は、JIS X 0201(片仮名図形文字集合を最上位ビットを1にして8ビットで表現する場合)と JIS X 0208 と組合わせて利用できることです。その際、これまでに蓄積された JIS X 0201 のデータは変換不要で、なおかつ JIS X 0201 と JIS X 0208の切り替えにエスケープシーケンスやシフト制御文字は要りません。

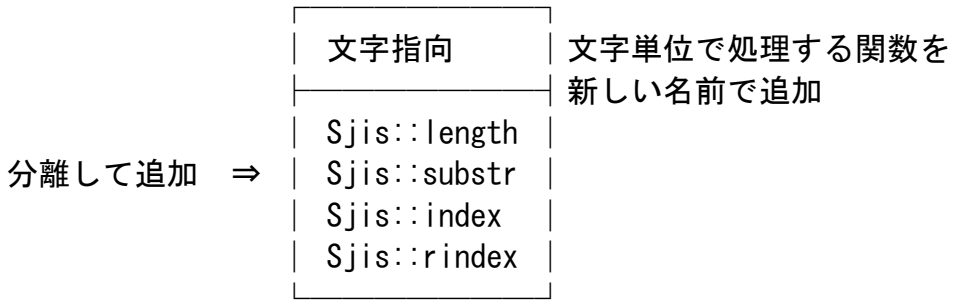
この上位互換性こそが ISO-2022-JP にも EUC-JP にも UTF-8 にもない、シフトJISだけの長所です。JIS X 0201 は当初 JIS C 6220-1969 という名称でしたが、40年経った今でもこのソフトウェアによって、そのまま利用することができます。

(2) スクリプトの上位互換性

今までのスクリプトは期待通りに動作します！

これまでのバイトおよび文字指向の関数、length, substr, index, rindex, pos はバイト指向の関数としてそのままの名称でそのままの機能で残し、文字指向の関数を新しい名称で分離して追加しました。このアイディアは AWK の日本語対応版、JGAWK を参考にしています。





なお、pos の文字指向版(Sjis::pos)は存在しません。

バイト指向の関数の引数にマルチバイト文字列を渡した場合、文字の位置やサイズはこれまで通りバイトで数えます。このソフトウェアによって length や substr が置換わることはなく、perl が提供する組込み関数をそのまま利用します。index と rindex はそれぞれ Esjis::index と Esjis::rindex に置き換わります。これらの関数は文字列の検索を文字単位で行いますが、その結果(文字列の見つかった位置)をバイト単位で返します。

マルチバイト文字列を「文字単位」で処理するのであれば、追加された文字指向の関数を使って記述する必要があります。既存のスクリプトが index, rindex を使っておらず、同様の処理を正規表現で行っていた場合はこの書き換えをせずに済むでしょう。

(3) 状態を持たない文字列

なので Latin-1 は使えません！

文字列が状態を持つ符号化方式として ISO-2022-JP があります。この符号化方式は電子メールでの利用が主で、それ以外の用途にはほとんど使われていません。その理由として、文字列が状態を持つためプログラミングが難しいことが挙げられます。

シフトJISをサポートし、なおかつ ISO/IEC 8859-1(Latin-1) をサポートするのであれば、0x80 から 0xFF の値のオクテットはシフトJISなのか Latin-1 なのかを区別して扱う必要があります。そうすると文字列が状態を持つことになり、何らかの変数が必要になります。変数を導入すると先に挙げた ISO-2022-JP と同じことになり、プログラミングはとても難しくなります。「実用化は不可能」と言うと言い過ぎでしょうか？

「Latin-1 のサポート」か「状態変数の導入阻止」かの選択において、Sjisソフトウェアの場合は後者を選びました。Latin-1 が扱えなくなるのを承知して、アプリケーションプログラマの負担を小さくすることを選んだのです。

■ 調和するソフトウェア

プログラミングPerl 日本語版の原著者のことばとして、Larry Wall さんは1977年型の

ホンダアコードに20年乗っている話がでてきます。そして希望通り、Perl はその車のように20年以上経ったいまでも世界に愛され続けています。

しかし、よく考えてみると初代のアコードと比べるのであれば、バージョン1 の Perl と比べるべきなので、寿命としてはアコードの勝ち、ということになるのかも知れません。あるいは現在リリースされている世代を比較するのであれば、Perl は 5 ないし 6代目なので、8代目のアコードがこれまた勝ちです(やれやれ)。

--- もうアコードと競うのはやめておこう

きっとハードウェアとソフトウェアを比べるのことが間違いなのです。そもそも太平洋を渡った最も価値あるものは、経済活動とは無縁のもののはずでした。

--- 例えば合気道のように

Perl は世界一を自認して、勝気になって競争し続けるあまり、負けてしまったようにも思えます。

なぜそうなってしまったのでしょうか？

プログラミング言語は競争する類のものではなく、むしろたくさんの異質なものをひとつにまとめる力が大切なのではないでしょうか？ Perl はもともとはそういったグルー言語としての能力が優れていました。しかしながら、近年は糊の力が弱くなり、Perl がプログラマの心から剥がれ落ちていくのを見かけるのです(とても残念でなりません)。

初めてのPerl の初版を読み返すと当時のPerlはプログラマの心をうまく捉えようとしていたと思います。

> コンピュータ科学者の中には(特に還元主義者たちは)否定する者もいるのだが、人間というものは変わった形の心を持っているものなのだ。心の形は平坦ではないので、ひどく歪めることなしには平らな面に写像することはできない。しかし過去20年以上にわたって、コンピュータ還元主義者たちは直行性の神殿に跪き、次いで立ち上がると、彼らの信奉する禁欲的な清廉さを誰かれとなく説いて回った。
>
> 彼らの熱烈だが見当違いの願望は、君たちの心を彼らの思考様式に合うように作り変えて、君たちの思考パターンをある種の超越次元の平原に押し込めることだった。平面に押し込まれるということは、まったく喜びのない存在になってしまうことに等しい。
>
> --- 「初めてのPerl」xviii より

幸いにも Perl は古い機能も新しい機能も持ちあわせているため、今では古いと考えられているシフトJISをサポートすることもできます。実際、Perl に新しい機能しかないの

だったらこのソフトウェアを書くことはできなかったでしょう。

シフトJISをサポートすることによって、Perl は再び私たちの心に近づき、私たちが直面している身近な問題を処理できるようになったのです。

Sjis ソフトウェアが目指しているもの --- それは Accord(調和) です。

・古いデータと新しいモジュールの双方に調和すること
・解決すべき問題とプログラマの心の双方に調和すること

そしてもっと調和するためにあなたの協力が必要です！

ぜひ以下のサイトを訪問して、JPerl の未来形 "Sjisソフトウェア" の進化にご協力ください！

http://search.cpan.org/dist/Sjis/

(投稿日付: 2009年12月6日;改訂: 2009年12月12日)

やｎでレPython
～λの使徒編～

Second Assistant　There's a `Bridget - Queen of the Whip'.
Nid　Yes...

二番目の店員　『ブリジェット　～鞭の女王～』、ありますよ☆
ニッド氏　そうだなぁ……

『Monty Python's Flying Circus "Tudor jobs agency"』

■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):ヤンデレパイソニスタ
・錦織武流(にしこり　たける):マナブの親父。システムエンジニア

第六回「リハなんてない毎日だから」

「マナブくん、今日の予定は？」
「えっと……バイトかな？」
「そう、じゃ、今日もダメだね」
「え、えっと、明日！　明日はバイト無いから！」
「ごめん、明日あたし用事あるんだ」
「う……」
　しばらくこの調子で予定がかみ合わず、マナブとイチコの午後の甘い語らい（視線は主にディスプレイだが）はお預けになったままだ。
　理由はマナブがバイトを始めたこと。
　毎年、ゴールデンウィーク前のこの時期にやっていたバイトということと、今年はイチコと知り合った関係から「何があってもいいように」軍資金を蓄えておきたいという「ちょっとした下心」があったからだ（もちろん、後者の理由はイチコには伏せてある）。
　とりあえず妙な誤解をうけないよう、きっちりバイトしていることをアピールするため、考えつく限りのあらゆる手を打った。
　バイト予定表（バイト先から正式に貰ったもの）をイチコに見せたり、バイト先（マナブの父親の会社）に許可を貰って、イチコにバイト風景を見学させたりしていた。
　なお、その際、イチコは件のマナブの父、武流(たける)と期せず対面することになった。
　すぐに二人の会話は、マナブの到底ついていけない次元に昇っていった。
　いたくイチコを気に入ったタケルは、イチコにもバイトに入るよう勧めたが、「バイト料はマナブの10倍出そう」という、マナブのプライドを粉々に砕く提示をしたにもかかわらず、「忙しい」との理由で断られた。
　「息子を宜しく頼みます」という、なんだか逆パターンの懇願をされて、複雑そうながらもなんだか嬉しそうなイチコだった。
　と、そんな経緯があるので、イチコとしても無碍には反対することもキレる（?）こともできずにいた。
　そんなストレスが、もしかすると後押ししたのかもしれない。
　イチコは、考えあぐねたあげく、口を開いた。

「ど、土曜……」
「え？」
「土曜日は、あ、あ、あいてる？」
マナブは絶句した。
講座は放課後、がセオリーだった。
別に示し合わせて放課後限定にしたわけではないのだが、いつのまにかそれが定番となり、それ以外がなんだか気まずくなるような雰囲気ができつつあったのだ。
しかし今回の件は、考えてみれば現状打破の格好の「口実」になることに気づいたのだ。
「も、勿論。イチコさんさえ良ければ」
「う、うん。じゃ、土曜の午前十時頃」
「よ、用意して待ってるよ」
少なくとも「でこちゅ」はクリアしてるのに、いつまで経っても初心な二人であった。

そして土曜日。
押された呼び鈴に一々過敏に反応しつつ、待ちに待った待ち人は……
……黒い人形だった。
(ご……ごすろり？)
確かに小柄色白で長い髪のイチコは、ゴシックロリータ風ドレス用マネキンといっていいほど似合いすぎている。
しかし、普段の理系っぽいイメージの性格からは、どう考えてもかけ離れている。
未だ見ぬ私服姿をあれこれと妄想していたマナブの想像を一気にぶっとばす、超巨大衝撃波だった。
「……マナブくん？」
「え、ああ……上がって」
いつものようにイチコを先に二階に上がらせて、マナブは紅茶を用意しながら、どう反応するか頭の中で整理していた。
そして、トレイを持って二階に上がる。
椅子（イチコ専用に持ち込んである）に腰掛けたイチコは、いつも以上にマナブの部屋から浮いていた。
「イチコさん、普段そういう格好してるの？」
すると、イチコの顔が少しだけ赤くなる。
「そんなことないよ。ただ、お姉ちゃんに相談したら、コレ着ていけ、って」
ふと、あることに気づく。
「えっと、僕はあんまり服装には詳しくないけど……それ、コスプレ用じゃなくって『本物』？」
「……うん」
コスプレ用のドレスは、どうしても布地が安っぽいものになりがちなのだが、近くで見るイチコの衣装は、間違いなく本物のしなやかな生地が使われていた。
「……変？」
マナブは首を振る。
「むしろ、似合いすぎ、で怖い」
「？」
首を傾げるイチコ。
「悪いオニーサンにお持ち帰りされそうで……」
イチコはくすぐったそうに笑う。
「マナブくんも、お持ち帰りしたくなる？」
「そんなことしなくても、デリバリーサービスで来てくれるでしょ？」
「あ、そっか」
イチコは自分の服を見下ろして、

「マナブくんがいいって言うなら、いいかな。実は家に、こういうの一杯あるから」
「そうなの？」
「うん。ホントはお姉ちゃんの中学生に上がる前のモノなんだけど……」
(やばいっ！)
　自分の肉体へのコンプレックスで落ち込みそうになるイチコを、マナブはさりげなくフォローする。
「可愛い服がずっと着られるなんて、得したね」
「ホントにそう思う？」
「僕は嬉しいよ」
　ふと、じと目でマナブを見上げる。
「マナブくんって……ろりこん？」
「イチコさん以外、眼中に無し」
　即答できたのは、そのうちくるだろうと思って用意しておいた科白なので。
　備えあれば憂いなし。

「今日はシークエンス(sequence)の勉強だよ」
「シークエンス？」
「うん。シーケンスでもいいけど。『連続した一塊』っていう意味。今までだと文字列(string)はシークエンスの一種だよ」
「文字の連続？」
「文字列はね。今日最初にやるのは、文字に限らず、数字でもなんでも配列できる『タプル(tuple)』だよ」
「タプル？」
「要するにオブジェクトのセットのこと。まずは見てみて」

```
>>> t = 0,1,2,"end"
>>> t[0]
0
>>> t[1]
1
>>> t[2]
2
>>> t[3]
'end'
>>>
```

「カンマで区切ってるだけみたいに見えるけど」
「そうだよ。基本的にはカンマに区切られたものはみんなタプルなんだよ。例えば前に、複数代入やったよね？」

```
>>> a, b, c = 1, 2, 3
>>> a
1
>>> b
2
>>> c
3
>>>
```

「これも実際は、タプルを利用したものだよ」
「ふぅん」
「あと、関数の引数も基本的にタプルだから、Python では色んなところで使われてるんだよ」
「文字と同じで、インデックスで呼び出せるの？」
「うん。インデックスもスライスも使えるよ」
「スライスって？」
「途中の切り出し。前にやったけど、名前は言わなかったね」

```
>>> t = 0, 1, 2, 3
>>> t[2:3]
(2,)
>>> t[1:3]
(1, 2)
>>>
```

「ああ、この区切りのややこしいの……」
「思い出した？」
「うん」
　あまり良い思い出ではなかった。
「難しい？」
「わかるけどわかりにくい」
「そう。ソレは慣れるしかないかも。でも、タプルみたいなシークエンスはいろいろ便利なんだよ」
「そうなの？」
「たとえば、ひとまとまりの数値があって、その全部の数値の２倍を表示したければ、一気にできるんだよ」

```
>>> t = 100, 200 ,300
>>> for x in t:
...   print(x * 2)
...
200
400
600
>>>
```

「for?」
「うん、for 文っていうループの新しい構文だよ。限定された回数だけ繰り返す時使われるんだよ。for の後の変数、今回は'x'だけど、これに次々 in の後におかれたモノに指定された要素を代入していくんだよ。今回タプルは順番に要素の並んだシークエンスだから、要素が順番に代入されていくんだよ」
　ふとイチコの口調が曖昧なのに、マナブは気づく。
「もしかして、シークエンス以外に、順番に要素を代入しないものもあるの？」
　イチコは悪戯っぽく笑う。
「気づいた？　そうだよ。複数の要素を含むオブジェクトをコレクション(collection)っていうんだけど、その中には順不同の集合とか、辞書なんてのもあるんだよ。ほかにもコレクション以外で、『次にどんな要素が返されるか』を規定しているだけの『反復子』なんてのも for 文で使えるんだよ。ただ、話が複雑になりすぎるから、今回は取り出す順序がはっきりとわかるシークエンスに話を限定するね」

「了解。ところで、出来た数値をタプルで返すことはできないの？」
「できるけど、どっちかというとそれは、リスト向けの話題だね」
「リスト？」
「うん。タプルと同じで順序で並んだ要素を扱うシークエンスなんだけど、タプルが基本的に要素の変更が出来ないのに対して、リストは要素の変更や追加が可能なんだよ」
「そうなの？　じゃぁ、リストだけあれば、タプルは要らないんじゃないの？」
「そうだね。確かにそう言う人もあるよ。でも『変更の効かない』ことがかえって便利なこともあるんだよ。どんな時かを説明するのは今は難しいけど、リストの話をしたときにちょっとだけ話すよ」
「ふーん。じゃぁ、数値をまとめて返すのも、リストの時だね」
「ううん。リストを使わなくてもできるよ。そうだ、マナブくん、考えてみてよ」
「え？　ちょ、ちょっと思いつかないんだけど……」
「そうだね、今のままじゃ情報不足だね。じゃ、タプルの性質をいくつか。まず、タプルどうしは'+'記号で連結できるんだよ」

```
>>> (1,2,3) + (4,5,6)
(1, 2, 3, 4, 5, 6)
>>>
```

「このとき、タプルをちゃんと括弧で囲っておかないと、'+'のほうが優先順位が高いから、ヘンな結果になるから注意してね」

```
>>> 1,2,3 + 4,5,6
(1, 2, 7, 5, 6)
>>>
```

「なるほど。'3+4'を先に計算しちゃうんだね」
「そういうこと」
「そういえば、文字列も'+'で連結できたよね。もしかして、タプルでも'*'のくりかえしが使える？」
「うん」

```
>>> (1, 2, 3) * 4
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
>>>
```

「あと、空タプルと要素１個のタプルの書き方だけ、ちょっと変わっているから覚えてね」
「空タプル？」
「要素が一つも無いタプルだよ」
「……変更できないのに、要素の無いタプルって、意味あるの？」
「さぁ？」
　意味ありげに笑うイチコ。

```
>>> ()
()
>>> 1,
(1,)
>>>
```

「空タプルは括弧だけの'()'、要素１個のタプルは、要素の後にカンマをつければいいんだよ」

「……カンマがタプルなんだよね？」
「うん」
「空タプルはなぜ、カンマが無いの？」
「理由は無いよ。ただ、そう決まってるだけ。ちょっと慣れないと判りづらいかもね」
「括弧をタプルの条件に……すると、計算の時、面倒か」
「勿論、理由を考えようとすればいくらでも考えられるけど、『決まってる』って言い切った方が潔いし、実際、簡単だから覚えやすいと思う」
「そうだね」
「じゃ、そういうことで」
「え？」
「タプルの数値を全て２倍したタプルを返す関数、書いてみて」
「か、関数にするの？」
「勿論、中でfor文を使えばいいだけだけどね。せっかくだったら関数の方が使いでがあるよね」
「う～、考えてみるよ」
（ひとつずつ作ったものを足していくんだから、連結を使って……ああっ！　階乗の時のアレが使えるか……、でも……あれ？　最初のタプル、どうしよう？　う～ん……あ、も、もしかして空タプルって……）

```
>>> def bai(t):
...     r = ()
...     for x in t:
...         r += (x * 2,)
...     return r
...
>>> t = 100, 200, 300
>>> bai(t)
(200, 400, 600)
>>>
```

「はい、ご名答。よく思いついたね」
「'x * 2'をそのまま足して、失敗したりとかしたけどね。でも、空タプルって、タプルどうしの計算の時に、最初に入れておくには確かに便利だよね」
「そうそう。あと、'+='はよく気づいたね」
「'r = r + (x * 2,)'を思い出す方が時間が掛かったよ。で、コレで通るなら、'+='も通るかな、って」
「悪くない連想だよ」
「もしかしてこの方法って、文字列にも使える？」
「使えるよ。じゃ、文字列を取って逆順にする関数、作ってみて」
「ぎゃ、逆順？」
「発想の転換だよ。ヒントは、'+='は使えない」
「う～ん」

```
>>> def gyaku(s):
...     r = ""
...     for x in s:
...         r = x + r
...     return r
...
>>> d = "DRACULA"
```

```
>>> gyaku(d)
'ALUCARD'
>>>
```

「うんうん。慣れてきたね、マナブくん。じゃ、タプルの要素の数を全部数え上げる関数……」
「len 関数が使えるんじゃない？」
「う……気づいてた？」
「確か、文字数を数える関数かって聞いたら『それだけじゃない』とか言ってた覚えがあるから」
「よく覚えてたね。あたしは忘れてたよ。でも、正解。len 関数は本来、シークエンスなんかのコレクションの要素の数を数える関数なんだよ」

```
>>> t = 100, 200, 300
>>> len(t)
3
>>>
```

「じゃ、それはそれとして、カウントする関数を『練習として』作ってみるね」
「……マナブくん、可愛くない……」

```
>>> t = 100, 200, 300
>>> def myLen(s):
...   r = 0
...   for x in s:
...     r += 1
...   return r
...
>>> myLen(t)
3
>>>
```

「ふと思ったんだけど」
「何？」
　イチコは若干膨れ気味である。
　キレずに不機嫌になる、というのは、それはそれで人間関係の成長なんだろう、と、マナブは前向きに考えることにした。
「文字を逆転する関数もありそうだよね」
「無いよ」
「そうなの？」
「無いよ。少なくとも Python の組み込み関数で、逆文字を返す関数は無いよ」
「じゃ、シークエンスを逆転させる関数も？」
「'reversed'っていう関数があるけど、戻り値が元のシーケンスじゃなくて特殊だから使いにくいよ」

```
>>> d = "DRACULA"
>>> s = ""
>>> reversed(d)
<reversed object at 0x009A8F70>
>>> for x in reversed(d):
...   s += x
```

```
...
>>> s
'ALUCARD'
>>>
```

「さっき言った反復子っていう種類の特殊なオブジェクトを返すんだよ。for で使う専用のシーケンスの一種、と思えば大体間違いない」
「素直にタプルかリストで返せばいいのに」
「前はそうしてたんだよ。でも、そういう『逆転』には時間がかかったりメモリを大量消費したりとパフォーマンスが悪いから、この形式になったんだよ。反復子の採用は、Python の中で例外的に『理解しやすさ』よりパフォーマンスを選んだ例だよ」
　やっぱり、少し深いところになると色々難しそうだなぁ、と、マナブはこっそり嘆息した。
「タプルはまだまだ色々あるけど、あまり深くやっても難しいから、次の『リスト』にいくよ」
「その前に休憩要求」
「そうだね」
「お茶、淹れ直してくるよ。あとお茶菓子も」

　そろそろ日差しが強くなってきたので……
「水羊羹？」
「うん。涼しそうでしょ？」
　冷緑茶の茶請け、である。
「ところでマナブくん、今、伝票入力のバイトやってるよね」
「うん」
「ああいうデータは、表計算ソフトかデータベースで操作するのが普通だけど、Python みたいな言語からも操作できるんだよ」
「そうなの？」
「うん。データベースに直接アクセスする方法もあるけど、個人で使うくらいなら、ＣＳＶが便利だよ」
「ＣＳＶ？」
「Comma Separated Value って言って、カンマでデータを区切っただけのテキストなんだけど、表計算ソフトからも読み込めるし、Python みたいなプログラミング言語でも扱い易いんだよ」
「そうなんだ」
「今日の後半の『リスト』は、データ操作の基本だよ。それとオブジェクト操作の、ね」
「オブジェクトって、数字や文字だっけ？」
「うん。でも、プログラミングの世界では『オブジェクト』っていう名前は色々に使われているから注意が必要だよ。今回は『オブジェクト指向プログラミング(Object Oriented Programming OOP)』のオブジェクトだよ。Python のオブジェクトは、OOP 的な意味でも『殆ど』オブジェクトであるとも言えるんだよ」
「そもそも OOP ってなんだか解らないけど、『殆ど』って何？」
「話せば長くなるけど、OOP には様々な解釈があって、その中には Python オブジェクトが要件を満たさないものもあるんだよ。だから『殆ど』。OOP が何かは、今のところは知らなくていいと思う。ただ、OOP で用いられるテクニックの中には、Python で使えるものも沢山あるってこと」
「ごめん、よくわかんないや……」
「いいよ、今は耳にいれておくだけで。でも、知らないなりに聞いておくのもいいと思うよ。プログラムやってると、そのうち聞く言葉だから」
「そうなの」
「うん。それじゃ、再開しようか」

トレイが片付けられ、講座再開。
「それじゃ、リストだよ。タプルで出来たことは大概リストでも出来て、リストは内容が変更できるんだよ」

```
>>> L = [10, 11, 12]
>>> L
[10, 11, 12]
>>> L[0]
10
>>> L[1]
11
>>> L[2]
12
>>> L[0] = 50
>>> L
[50 ,11 ,12]
>>>
```

「なるほど、インデックスで代入できるんだ」
「うん。今まで扱ってきたオブジェクトは全部『不変(immutable)』だったけど、リストは『可変(mutable)』オブジェクトだよ。可変オブジェクトの取り扱いには注意が必要だよ。間違えると、オブジェクトを壊しちゃうからね」
「こ、壊す？」
「うん。*もしマナブくんがあたしを見てくれなくなったら、あたしが壊れるみたいに*……」
久々に、イチコの目に狂気の色が浮かぶ。
マナブは速やかに『防衛プログラム』を開始。
「それは『ありえないこと』。今は『壊れる可能性』のあることだから、例が不適切だよ、イチコさん」
「本当？」
「疑うなら刺せばいいよ」
刺さない常識を持たない人間に此れを言うのは命懸けだが、同じ命懸けなら少しは格好つけさせてもらう。
……とか頭で考えながら、微かに足が震えるマナブだった。
「マナブくんを信じる」
とりあえず、当座の危機は去ったようだ。
「で、可変オブジェクトは？」
「うん。今までの、ある決まったオブジェクトに、『代入』っていう操作で、名前をつけてきたよね。でも、オブジェクトそのものは変化して無かったよね？」
「文字列やタプルの連結は？」
「あれは『連結して出来た新しいオブジェクト』に名前をつけただけだよ」
「オブジェクト自身変化しなかったってことだよね」
「うん。その点、リストの要素への代入は、オブジェクトそのものの内容が変化するんだよ」

```
>>> x0 = 0
>>> x1 = x0
>>> x0 = 1
>>> x1
0
>>> y0 = [0]
```

```
>>> y1 = y0
>>> y0[0] = 1
>>> y1
[1]
>>>
```

「……えっと、ちょっと混乱ぎみ」
「うん。じゃぁ、少し詳しく説明するね。画面にはでてこないけど、オブジェクト自身をそれぞれ『整数 0』、『整数 1』、『リストA(内容)』としてみるね。代入で、変数がどのオブジェクトを指しているかを一つ一つ追っていくと、こうなるよ」

```
x0 = 0        x0 → 整数 0
x1 = x0       x1 → 整数 0
```

「x1は右辺のx1が『評価された結果』が代入されるから、x0自身じゃなくって、『整数 0』が代入される。これはＯＫ？」
「うん。今まで通りだね」

```
x0 = 1        x0 → 整数 1
```

「再代入は単なるラベルの付け替えで、x1には影響を及ぼさないから……」

```
x1            x1 → 整数 0
```

「……と、ここまではＯＫ」
「エス、マム」
「だけど、ここからが本番」

```
y0 = [0]      y0 → リストA
              リストA[0] → 0
```

「え？　二つ一緒？」
「うん。リストが生成された時、同時にリストの要素も内容を指すんだよ。Pythonはこれを一気に書いてるけど、プログラミング言語によっては、二段階で設定する必要があるものもあるよ」
「書くのは簡単だけど、解りづらいね」
「そうかも。本当は意識しなくてもいいように設計されてるからだと思うけど、ここは基本だから」
「イチコさんを信用してるよ」
「うん」

```
y1 = y0       y1 → リストA
```

「ここは同じなんだね」
「うん。違うのはここから」

```
y0[0] = 1     リストA[0] → 1
```

「え？」

「不思議？」
「うん」
「じゃ、順を追って説明するね。これも実は、二段階で評価されてるんだよ。まず、y0 が評価されてリスト A になり、そのリスト A のインデックス０に１が代入されたというわけ。だから……」

y1[0] y[0] → リスト A[0] → 1

「……となるわけ」
「うぁ、解り辛い……」
「ふふふ。インデックスを'+'や'-'みたいな演算子だと思えば簡単だよ」
「え？」
「つまり、ａに５、ｂに６が入ってて『ａ＋ｂ』するときは、それぞれａとｂがまず評価されて５と６になってから『５＋６』するよね？　だからこの場合も、[]っていう演算子を使用する演算として見るんだよ」
「まぁ、そう言われれば……」
「これから、複雑な式を書くことになると思うけど、そのときには、オブジェクトの式の評価の順番をちゃんと理解してないとおかしくなるよ。基本的には、『遅延評価』って言われる特殊な処理をしないもの以外は、オブジェクトは全部書かれた時点で評価されると思えばいいよ」
「遅延評価って？」
「関数みたいに、後から値が入るようなもの、かな？」
「なるほどね」
「関数の名前が出たところで、解りづらいバグの原因になるリストと関数の例をみてみるね」

```
>>> l = [0]
>>> m = [0]
>>> n = 0
>>> def change(x, y, z):
...   x[0] = 1
...   y = [1]
...   z = 1
...
>>> change(l, m, n)
>>> l
[1]
>>> m
[0]
>>> n
0
>>>
```

「関数の中での操作は、外に影響しないんじゃなかったの？」
「変数は共有しないけど、オブジェクトは共有してるから、再代入には影響をうけないけど、オブジェクト自身の変更は当然有効なんだよ」
「なんとなくわかったような、わからないような」
「あとは、自分で書いてみることだよ。それに、考えようによっては、オブジェクトの中身を操作する手続きをパッケージ化できるから、便利な面もあるんだよ。『参照渡し(call by reference)』って言ってね、たとえば、リストの中身を全部２倍する関数とか、書けるよね」
「……あ、そっか。リストは変更可能だから、内容の更新ができるね」

「そういうこと。書いてみる？」
「『よろこんでっ！』」
　悪乗りまでしてディスプレイに向かったマナブだが、すぐに先程の例がそのまま使えないのに気づく。
「要素、は整数だよね？」
「そうだね（くすくす）」
「整数は不変だよね？」
「うんうん。気づいた？」
　例によって、引っかけ、である。
「マナブくん、降参？」
「う～ん、それはそれで悔しいなぁ……そうだ、len関数ってリストでも使えるよね？」
「うん」
「じゃ、なんとかなるかも……」
「え？」
　マナブは、すでに画面に向かってコードを打ち込んでいた。
「えっと……あ、インデックスは0からだっけ。そうするとココで最初に減らしておいて……」

```
>>> def bai(L):
...     n = len(L)
...     n -= 1
...     while n >= 0:
...       L[n] = L[n] * 2
...       n -= 1
...
>>> l = [100,200,300]
>>> bai(l)
>>> l
[200, 400, 600]
>>>
```

「できたできた。最初、for文にこだわってたから難しかったけど、while使えばできるじゃん」
「……」
「あれ？　どうしたの、イチコさん？」
「これ、あたし、思いつかなかった……」
「へ？」
「ううん。使ってる言語がPython じゃなかったら、多分頭を切り替えてたけど」
「どういうこと？」
「マナブくん、プログラマはね、使用しているプログラミング言語によって、思考方法を左右されるんだよ」
「Pythonを使えばPython流、ってこと？」
「うん。それぞれのプログラミング言語特有の手法があるから、それをどう使うかっていう方向に頭が働くんだよ。逆に、あまり使わない方法は、例え可能でも、わからない場合もあるんだよ」
「イチコさんなら、どう書くの？」
「'range'っていう、殆ど文法の一部になってる関数があるから、それを使うんだよ。range関数は反復子を返す関数で、一番簡単な使い方は、整数を指定すると、0から順にその整数個数、つまりその整数-1の数まで整数を生成するんだよ」

```
>>> for x in range(10):
...     print(x, end=", ")
```

```
... else:
...   print()
...
0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
>>>
```

「そっか、つまりコレに、リストの長さを入れてやれば、インデックスを順に指定してくれるんだね」
「うん」
「だったら、使い方はこうかな？」

```
>>> def bai2(L):
...   for x in range(len(L)):
...     L[x] = L[x] * 2
...
>>> l = [100,200,300]
>>> bai2(l)
>>> l
[200, 400, 600]
>>>
```

「あ、コレいいね。カウンタ使うより判りやすいし」
「うん。でも、その代わりにカウンタの使い方を思いつかなくなった」
「イチコさん？」
　気がつくと、イチコは自分の体を抱き締めて小さく震えていた。
「人間はやっぱり、楽するとダメなんだ。柔軟さが段々失われていく……」
「そ、そうかなぁ」
「こんなのじゃダメだよ。こんなのじゃ、マナブくんを教え続けられない」
　あちゃ、とマナブは頭を抱える。
　どうやらネガティブスパイラルに陥ったらしい。
　さて、この困ったお姫様をどう宥(なだ)めるか……
「じゃ、ちょうど良かったね」
「え？」
「イチコさんが忘れてたこと、僕を教えることで思い出したんでしょ？」
「う、うん」
「ということは、僕はイチコさんの役に立ったってことだよね？」
「え？　え、えええっ！？」
　にっこり笑うマナブは、イチコが何かの言葉を返す前に、畳み掛けるように喋る。
「ほら、僕はイチコさんに負担かけてばっかだと思ったけど、僕を教えることでイチコさんの脳トレになったなら、それはラッキーだよね」
「そ、そうかも」
「じゃ、僕は遠慮なくイチコさんに甘えて教えてもらってもいいわけだよね」
「う、うん」
「じゃ、せんせー、次お願いします」
「も、もうっ、マナブくんったら……」
　思わず笑い出すイチコに、マナブはこっそり心中でピースサインを出した。
「じゃ、次にいくよ。リストに要素を加える操作」
「え？　インデックスに代入していけばいいんじゃないの？」
「やってみて？」

「うん」

```
>>> l = [0,1,2,3]
>>> l[4]=4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>>
```

「あれ？　エラーが出た」
「存在しないインデックスに代入しようとしても、ダメなんだよ」
「じゃぁ、リストの長さは不変なの？」
「そんなことないよ」
「あ、そっか、連結を……」
「それ、タプルでもできるでしょ？　新しくリストを作るんじゃなくて、今のリストを延長すること、ちゃんとできるよ。見ててね」

```
>>> l
[0, 1, 2, 3]
>>> l.append(4)
>>> l
[0, 1, 2, 3, 4]
>>>
```

「……な、なんか見慣れない形式のモノが……」
「気づいた？」
「うん。ドットの前の'l'って、多分このリストだよね？」
「そうそう」
「ドットで繋ぐと、どうなるの？　append は関数っぽいんだけど……」
「append は関数の一種だけど、『メソッド』っていうんだよ。メソッドは、オブジェクトの『機能』を引き出す方法で、今回はリスト自身が持つ『要素を加える』っていう機能を呼び出して使ったんだよ」
「リストの機能？」
「そう、append はリストという型に予め設定されている機能なんだよ。オブジェクトという名称は、こういった機能を内蔵したものを指す場合があるんだよ。そして OOP つまりオブジェクト指向プログラミングは、オブジェクトのメソッドを使ったプログラミングなんだよ」
「……えっと……」
　まなぶはこんらんしている（古いＲＰＧ風）
「う〜ん、ちょっと難しいかな。でも、Python の機能はかなりの部分がこのオブジェクトのメソッドに隠されてるから、理解してほしいな。じゃ、順を追ってゆっくり説明するから、ついてきてね」
「らじゃ」
「まず、関数との関係。ホントは関数もオブジェクトなんだけど、話がややこしくなるから、ここでは別物として考えるね。関数を使ったプログラミングの基本はこんな感じだよね」

```
        func(object)
```

「うん」
「オブジェクト指向プログラミングの基本は、こんな感じだよ」

```
object.method()
```

「どうして形式を統一しないのかなぁ」
「それは、プログラミング言語の流儀かな。例えばお姉ちゃんが好きなLisp系の関数型プログラミング言語では、全部関数適用の形になるよ。それに対してSmalltalkっていうオブジェクト指向プログラミング言語の元祖みたいな言語は、全部メソッドを使うんだよ。あ、Smalltalkでは『オブジェクトのメソッドを呼び出す』とは言わずに『オブジェクトにメッセージを送る』って言うんだけどね」
「統一してる言語はあるんだよね。で、Pythonは両刀？」
「うん。言語で言うと、C++系かな。統一してある言語は、フォーマットが単純で、それぞれの得意分野については威力を発揮するけど、反面、苦手分野では、書き方が若干回りくどくなるんだよ。両方採用の言語は、両方の知識が必要だけど、シチュエーションによって方式を選ぶことが出来るから、コーディングしやすくなる。一長一短だよ」
「それぞれの得意分野って？」
「たとえば、関数型だと扱うデータの型がシンプルだから、操作の手順の流れを追いやすいんだよ。反面、複雑なデータ構造を扱う場合は手順がやや煩雑になるんだよ。OOPは、操作をブラックボックス化するから、大きな流れを必要なところだけ追うのに適してる。それに、オブジェクトに加えることができる操作が限定されるから、できることとできないことがはっきりしやすい。反面、ブラックボックスになってる部分の挙動を確認するのが難しかったり、プログラミングでできることに大きな制限がかけられたりすることがあるんだよ。極端な話、明快さと手軽さのトレードオフかな。関数型はマニュアル車で、オブジェクト指向はオートマ車みたいな……」
「イチコさん」
「何？」
「ごめん、説明長くてわかんない。実践で覚えるよ」
「……マナブくんはせっかちさんだね」
　イチコの説明を遮るのには若干の不安があったが、それ以上聞き続けてもマナブが理解できそうにもないことも事実だった。
「それじゃ、Pythonの話に限って言うね。Pythonでは色々なデータ型が最初から使えるし、やろうと思えば新しい型を自分で書くこともできるんだよ。メソッドは主に、可変型のオブジェクトの内容を変更する際に使用されるよ。不変型のタプルのメソッドは、ユーザが通常呼び出して使うのはcountとindexの二つだけど、リストにはその二つに加えて、append, extend, insert, pop, remove, reverse, sortなんかがあるんだよ。もっとも文字列は不変型としては例外的に、書式を整えて出力できるように、リストより沢山のメソッドを持っているけどね」
「タプルとリストに共通な、countとindexってどう使うの？」
「countは、引数として取ったオブジェクトが、要素として何個あるか『カウント』するメソッドだよ。indexは、引数として取ったオブジェクトが最初に現れるインデックスを返すんだよ。countは要素に指定されたオブジェクトが無くても0を返すだけだけど、indexは指定されたオブジェクトが要素に無いと、エラーを返すんだよ」

```
>>> t = (5, 4, 3, 2, 2, 1)
>>> t.count(3)
1
>>> t.count(2)
2
>>> t.index(2)
3
>>> t[3]
2
```

```
>>>
```

「append は要素を末尾に加えるメソッド、だっけ？」
「そうだよ。要素を途中に挿入するなら、incert が使えるよ」

```
>>> L = [500,400,300]
>>> L.append(100)
>>> L
[500, 400, 300, 100]
>>> L.insert(2,666)
>>> L
[500, 400, 666, 300, 100]
>>>
```

「extend は？　なんか『延長』っぽい名前だけど」
「extend はリストなんかのシーケンスをまとめて末尾にくっつけるメソッドだよ。普通はリストを使うけど、タプルでも、なんなら文字列でもいいんだよ。ただし文字列を使うと、１文字が１要素になっちゃうから注意が必要だよ」

```
>>> L
[500, 400, 666, 300, 100]
>>> L.extend([10,20])
>>> L
[500, 400, 666, 300, 100, 10, 20]
>>> L.extend((3,4))
>>> L
[500, 400, 666, 300, 100, 10, 20, 3, 4]
>>> L.extend("hello")
>>> L
[500, 400, 666, 300, 100, 10, 20, 3, 4, 'h', 'e', 'l', 'l', 'o']
>>>
```

「pop は？」
「pop と remove は要素を外すメソッドだよ。pop はインデックスで指定した要素を外して戻すメソッド。remove は要素を指定して、最初に現れた要素を外すメソッドだよ。pop の引数を省略すると、末尾の要素を外して戻すよ」

```
>>> L = [5,4,3,2,1,0]
>>> L.pop()
0
>>> L
[5, 4, 3, 2, 1]
>>> L.remove(1)
>>> L
[5, 4, 3, 2]
>>> L.pop(3)
2
>>> L
[5, 4, 3]
```

```
>>>
```

「pop と append はペアで、簡単なスタックを実装するのに使えるよ」
「スタック？」
「LIFO(last in first out)つまり、最後に入れた要素から最初に取り出すルールを持つデータ構造だよ。スタックっていうのは本来『書類の山』みたいな意味で、山積になった書類は上からしか、つまり新しく積んだものからしか取れない、っていうイメージでつけられた名前だよ」
「中途から取ると、山が崩れちゃいそうだしね」
「そうそう。リストの append メソッドを使うと、リストの末尾に新しい要素が加えられ、引数無しで pop メソッドを使うと、末尾のデータ、つまり一番最後に新しく付け加えられたデータから取り出せるんだよ」

```
>>> stack = []
>>> stack.append(100)
>>> stack.append(200)
>>> stack.append(300)
>>> stack.pop()
300
>>> stack.pop()
200
>>> stack.pop()
100
>>>
```

「面白いね。でも、コレ何に使うの？」
「プログラムコードの解析、かな。特定目的に使う小さな言語、例えば計算用のプログラミング言語を自作する時なんか、よく使うよ」
「プログラミング言語でプログラミング言語を作るの？」
「うん。よくあることだよ」
「それって、無駄に時間がかからない？」
「そうでもないよ。フルスペックの言語を作るのは膨大な時間がかかるけど、特定目的、たとえば図を描画するとか計算するとか、あるいはデータを整形してレポートを出力する言語とかなら、一々そういった作業に対応するプログラムを作るより、簡易言語を作ってしまったほうが簡単な場合があるんだよ」
　マナブには、まだまだ理解し難い世界が広がっているようだ。
「ところでイチコさん、打ち込んでて思ったんだけど、stack.append って、名前長くない？」
「ふふふ、そうだね。stack はともかく、append って一々打つの、面倒だよね。それにスタックは、普通'append'じゃなくって'push'って言うよ」
「メソッドの名前って、変えられないのかな？」
「変えられるけど、その方法はもう少し先にするね。今はもっと簡単な方法」
「へ？」
「プログラマが見て度肝を抜く、Python の必殺技だよ」

```
>>> STACK = []
>>> pop  = STACK.pop
>>> push = STACK.append
>>> push(100)
>>> push(200)
>>> push(300)
```

```
>>> pop()
300
>>> pop()
200
>>> pop()
100
>>>
```

「……」
「どう？」
「もしかして、STACK.pop に pop っていう『名札』をつけたの？」
「当たりぃ！」
　得意そうに笑うイチコ。
「長い名前、こんなに簡単に省略できるんだ」
「これから先、長い名前が出てきても、一度書けばあとは短い名前にリネームできるよ。もっとも、名前のつけ方は慎重にしないと、自分が混乱しちゃうけどね」
「そういえば、前に print を簡単にする方法があるとか言ってたけど……もしかして……」

```
>>> p("hello")
hello
>>>
```

「あはは、マナブくんが見つけちゃったね。Python では関数もメソッドもどちらもファーストクラスオブジェクト、つまり普通のオブジェクトだから、整数や文字と同じように、代入できるんだよ」
「ううう、知恵熱が……」
「じゃ、今日はこのくらいにしようか」
「段々難しくなってきたなぁ」
「慣れだよ。考え方の慣れ。マナブくんもそのうち、Python 流の頭の使い方に慣れてくるよ」
　ふと、『調教されているみたいだ』という意識が頭によぎる。
　イチコの顔を見て……
（調教、されてるんだろうなぁ、Python じゃなく、イチコさんに）
とか思うマナブだった。
（Python は失敗してもエラーが出るだけだけど、イチコさんに失敗すると……）
　あまり考えたくなかった。

【内容】
・タプル（インデックス参照、連結、繰り返し）
・リスト（インデックス参照／代入）
・メソッド
　タプルとリストの共用(count, index)
　リスト用(append, incert, extend, pop, remove,)

【注釈】
・**『λの使徒編』**：あ、アズサさん出すの忘れた（汗）
・**「リハなんてない毎日だから」**：アドリブ勝負、元気に Yeah! Yeah!……って、アイマス以来、ポリゴンの女の子のスゴいことスゴいこと。
・**マナブがバイトを**：手書き伝票の打ち込み。相変わらず、IT化してない顧客はある。
・**放課後**：なんとなくけだるいイメージがあるのは、私だけか？
・**でこちゅ**：おでこにチュ（第４回参照）
・**ごすろり**：ゴシックロリータ。色々話し出すと長いが、要するにアレ。
・**本物**：ブランド品、という意味ではない。コスプレ衣装ではなく、ちゃんとした衣服である、というだけの意味。
・**ろりこん(Lolita complex)**：本物のロリータコンプレックス＝ニンフェットマニアは、早熟な大人っぽい女の子が好みであって、ゴスロリドレスなんか着てるお菓子系（？）のお人形さんのような子どもっぽい女の子は興味は無い（本当）。だからルイス＝キャロルもロリコンではない。
・**シークエンス(sequence)**：シーケンスとも言う（発音的にはシークェンスくらい）。「配列」などと訳されることもあるが、Arrayと区別がつかないのでそのままカタカナ表記とした。Pythonでは文字列、タプル、リストなど、順列インデックスで呼び出せるコレクションをそう呼ぶ。
・**タプル(tuple)**：もと連結形で「～個の要素からなる(集合)」といったような意味。データベースの一件のデータ、といったような意味でも用いられる。Python的には不変(immutable)のリスト。
・**カンマで区切っている**：タプルの要件は、カンマで区切ってあることであって、パーレン'()'で括ってあることではない。
・**for文(for statement)**：必ずinを一緒に用いるので『for-in文』と言ったほうが適切かもしれない。本来、コレクションの要素を一つずつ取り出すための構文だが、反復子の登場で、正確な定義がしづらくなった。
・**コレクション(collection)**：要素のコンテナであるオブジェクトのこと。Python3以降、setなどますます充実してきている。
・**反復子(iterator)**：イテレータ、とカタカナで書くのが主流。言語によってどんなものかがかなり違う。Pythonでは、for文で用いてあたかもコレクションのように次々と要素を渡すオブジェクトを指す。for文を用いない場合は、next関数(Python3ではnextメソッドではなく関数に変更されているので注意！)を用いて順に要素を取り出すことが可能。要素が尽きると、StopIteration例外を投げる。使いきりで、一度『尽きた』反復子は廃棄するしかない。なお、イテレータの内部に本当に要素が蓄えてあるかは不明。ジェネレータ(generator)のように、要素をその都度生成しているものもある。
・**空タプル(empty tuple)**：要素０のタプル。この書式'()'が、タプルを誤解させるもとなのだが、多分Lisp以来の空リストの書き方のスタンダードに倣ったものと思われる。
・**"DRACULA"**：竜＝悪魔という意味……なんて言わずとも、誰でも知ってる言葉。
・**reversed 関数**：シークエンスを引数に取って、逆順の反復子を返す関数。Python2に慣れている方は、Python3では、反復子を返す関数がやたら増えているので注意(range, zip, map, functools.reduceなど)。そこで、反復子をシーケンスに直すイディオムを知っておくと便利

```
文字列の反転　srev = lambda s: ''.join(reversed(s))
タプルの反転　trev = lambda t: tuple(reversed(t))
```

・**反復可能オブジェクト(iteratable object)**：反復子に加え、コレクションやファイルなど、for文で使用できるオブジェクトの総称。iter関数に通すと、反復可能オブジェクトの反復子が得られる。
・**CSV Comma Separated Value**：コンマでレコードが区切られたデータ。ぶっちゃけ、もしCSVを操作

するだけなら、私は Python ではなく AWK を使う。
・OOP には様々な解釈があって：話によると三つくらいあるらしい。実はその３つの解釈のどれを採用しても、Python は OOP 言語としては不十分、という結果になっているような気がするのだが……
・不変オブジェクト(immutable object)、可変オブジェクト(mutable object)：オブジェクト自体が持つ値が変化しないオブジェクトと、変化可能なオブジェクト。なお、不変オブジェクトは辞書などのキーとして使えるが、タプルなどの場合は、その要素が全て不変である hashable オブジェクトである必要がある。
・参照渡し(call by reference)：代入に対して『箱に入れる』のではなく『名札をつける』と一貫して覚えておけば、Python の参照を理解するのは比較的簡単である。ただし、ここでいいかげんに覚えると、結構あとで苦労しそうな気がする（特に C 言語などを知っている人は余計に）
・『よろこんでっ！』：至極個人的な意見だが、私はこの掛け声が嫌いだ（なんか莫迦にされてるみたいでムカつく）
・while 使えばできる：同じ for 文でも、C 言語のようなものは基本的にカウンタを使うこちら。シークエンスを舐めるタイプの for(foreach)が使える言語は、使い出すとやめられない。蛇足だが、Python に慣れた人なら、こんなものは当然のように内包を使うだろう。
・range 関数：これも先に述べた反復子を返すタイプの関数に変更された。というより、多分このプロトタイプの xrange 関数が反復子を返す関数の先駆けだろう。
・メソッド：第六回目にして、初めてメソッドを使用。というか、今まで使わずに話を進めるのに、こっそり苦労していた。
・オブジェクトにメッセージを送る：厳密に言うなら、『object.method()』の形は、object というレシーバの'.'というセレクタに、二項メッセージ式で method を送り、その返り値に'()'というセレクタで単項メッセージを送る……とでも言うのが正しいのだと思われる。Smalltalk で Python を考えるのは難しい。
・関数型(FP)、オブジェクト指向(OOP)：FP から見ると、OOP は副作用のカタマリの恐ろしいモノに見えるだろうし、OOP から見れば FP は原始的でスマートさに欠けるように見えるのだろう。Python はこれに手続き型を加え、テキトーにコードを書く。
・ユーザが通常呼び出して使う：メソッド名の前後にダブルアンダースコ'__'のあるメソッドは、基本的には一般ユーザが使うものではない。ただし、Python の hack が趣味の人は、通常のメソッドと同様に使う。どんなメソッドが実装されているかは、dir 関数に掛けて調べてみると良い。
・プログラマが見て度肝を抜く：あまりのいいかげんさに……かも。

【おまけ】クロージャを使ったスタックを１行で書く方法……

```
stack=lambda item=None, s=[]:(None if s==[] else s.pop())if item==None else s.append(item)
```

【Python あれこれ】４．反復子の憂鬱

「反復子(iterator)の利用は Python を征服し、統一した」

『Python チュートリアル』より

Python2 から Python3 への移行によって、Python の文法はすっきりと整理され、概ねわかりやすいものとなりました。ただ一点、パフォーマンス面の改良によって導入された反復子を除いて。リストやタプルで返されるのを期待していた range や reversed で反復子を返されると、今でも若干の戸惑いと苛立ちを感じます。list 関数や tuple 関数（その実はコンストラクタ）に渡す一手間も、繰り返しになるとあまり気分の良いものではありません。「こういう〈儀式〉がイヤで Python を使ってるんだけどなぁ」とか、思わず愚痴が出たりします。実際、for 文のコレクション処理では、内部的に iter 関数と next 関数が呼ばれているらしいのですが、こういった水面下の変化ならばともかく、表立って登場されると、コレクションとは異なり要素表示もインデックス参照もできない（当然ですが）反復子は、特にインタープリタ上での作業の多い私にとっては実に鬱陶しい存在です。リストを逆転して返せと言ったたら、反復子を返すとか……死んだオウムを買わされたような気分です（まぁ、反復子は生き返りますけど）

今まで「プログラミング初学者に Python を」と勧めてきた立場としても、反復子の「征服／統一」の事実は頭の痛い問題です。プログラミングを学び始めた人には、できるだけ『儀式(idiom)』ではなく『魔法(arche)』を覚えて、自己流にぶん回して欲しいというのが私の方針です（『自由自在 Squeak プログラミング(ソフト・リサーチ・センター刊)』の巻頭言で Alan Kay が「ライブラリに頼るな」とか言ってたけど、似たようなモノかも）が、反復子に関してはどう見ても、最初はイディオム的に for 文で使うと覚える方が無難でしょう。もともと、range 関数を用いて回数指定する方法ですら私はあまり好きではなかったのですが（それについて PythonML で議論したこともある）range 関数が返すモノがダイレクトに見られないとなると、そんな魑魅魍魎を初学者に与えるのか（ご存知のとおり、for-in-range()は基本中の基本の構文）と思うとぞっとします。イディオムの暗記は、分析的思考の停止を意味しますので、あまり推奨できるものではありません（私が理科の授業で公式を暗記させないのと同じです）結局、イディオムとして登場させ、早期に反復子の原理を学ぶ、という方法が一番妥当だとは思うのですが、はてさて、どんな比喩が判りやすいでしょうかねぇ……

（機械伯爵）

（投稿日付: 2009年12月7日）

編集後記

jscripter

毎度、巻頭言を書いて、編集後記を書いている。実はまだ自分の記事は完成していなかったりする。なんて、いつも同じことを書いているが、今回はまだ自分以外の原稿も揃っていないところが違う。

3ヶ月毎に新しい記事を完成させるのは実はそれほど楽ではない。たとえ数ページの記事でさえ。なぜなら、刊行した翌月はまだ先のことだと思ってのほほんとしている。その翌月はまだもう一月あると思う。そうして、当月になって焦るということになる。

僕の場合は日記を書いているから、それを書くだけで日々は過ぎていく。スクリプトを書いている暇はそれほどない。スクリプトで必要なものは既に書いてしまったから、書く必要のあることも特にないと言えるかもしれない。そこで、何かネタを探すことになる。それは自分で興味があり、かつ生産的なものであらねばならず、さらには他人が興味を持ちそうなものである必要がある。

新しい発想が欲しいと思う。しかし、新しいものって何?

未完成のモノって、Perl 6/Parrotぐらいしかない。どんなふうに新しいのか知りたい。HTML 5/CSS 3、Google waveもかな。そして、もっと、もっと新しいもの・・・それは逆に古いものかもしれない・・・

(投稿日付: 2009年11月29日;改訂: 2009年12月12日)

ＴＳＮＥＴスクリプト通信　ISSN 1884-2798　出版地: 広島市

2009年12月15日　2.3.006版 改訂
2009年12月12日　2.3.004版 刊行
2009年11月30日　2.3.000版

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
海鳥　kaityo256 at nifty dot ne dot jp
ムムリク　qublilabo at gmail dot com
jscripter　jscripter9 at gmail dot com
稲葉 準　hinaba at hotmail dot co dot jp
機械伯爵　kikwai at livedoor 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 3.1.1 Writer

発行所

一次配布所: TSNETスクリプト通信刊行リスト
http://text.world.coocan.jp/TSNET/?TSNET%E3%82%B9%E3%82%AF%E3%83%AA%E3%83%97%E3%83%88%E9%80%9A%E4%BF%A1%E5%88%8A%E8%A1%8C%E3%83%AA%E3%82%B9%E3%83%88