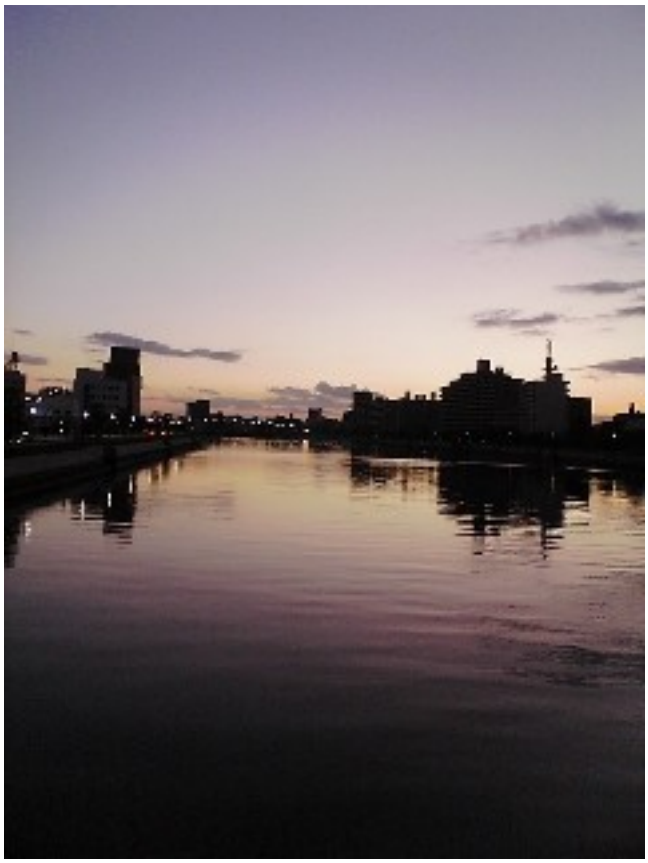
# TSNET スクリプト通信



TSC 編集委員会

# 目次

# 巻頭言

jscripter

TSNETスクリプト通信第4号刊行。昨年五月の創刊当時、季刊ならなんとかなるのではと考えていたが、ここまで辿り着けるかどうかは定かでなかった。今号で春夏秋冬号すべてを届けることができた。関係者のみなさんとともにまずは喜びたい。読者の方々の御蔭である。

今回は、思いがけず、海鳥(渡辺宙志)さんにRubyの記事を寄せていただき、第1巻第4号を飾ることができた。さらに機械さんの力作、「やｎでレPython ～新約蛇神祭祀書～」の第2回、Yささんの定番awkスクリプトゲーム「状態」(なんだか、哲学的)を掲載し、もちろん、海鳥さんの「よしおさんとロボ太」も連載、拙作「PerlMagick & Cooliris入門」を加えて、多彩で充実した構成となったと自画自賛している^^;)

また、今号からは「TSNETニュース」を掲載している。季刊なのですべてタイムリーというわけにはいかないが、大きな視点で重要な動きを取り上げていきたい。記事をお寄せいただければ大変ありがたい。

「よしおさんとロボ太」は、「メガネロボ」、「続・メガネロボ」、「ロボ動力」の三作を転載させていただいた。「ロボ動力」は新作である。このマンガの意味がわからない場合には、「シュークリーム分」でググること^^;)

表紙は「PerlMagick & Cooliris入門」の記事で使った夕暮れの風景写真である。広島の三角州に流れる京橋川の平野橋より御幸橋方面を望む。瀬戸内海はすぐ先にある。写真など、表紙に掲載するものも募集中です。

# TSNET ニュース 1.4

## Python 3.0 final リリース

Python 3.0 final が 2008 年 12 月 3 日にリリースされ、現在は 2009 年 2 月 13 日のバグフィックス・リリース、Python 3.0.1 が利用可能である。Bruce.さん、Fe2+さんの助力を得て訳した「What'sNewInPython3.0」は TSNET の人気コンテンツとなった。PyJUG の浦郷氏の紹介記事の御蔭が大きい。

Python 3.0.1 Release
http://www.python.org/download/releases/3.0.1/

TSNETWiki の「What'sNewInPython3.0」翻訳ページ
http://text.world.coocan.jp/TSNET/?What%27sNewInPython3.0

PyJUG の紹介記事
http://www.python.jp/Zope/PyLog/1232683984

Python 3.0 final のリリースを以て、Python の歴史は新しいフェーズに入った。賛否両論あるが、たぶんこの大きな変更は、Python が将来生き残る言語になるための試金石となるだろう。

(機械伯爵)

## Parrot 1.0 リリースは 3 月 17 日予定

いよいよ、Parrot 1.0 が出る。そろそろ関心のある人は動く時だろう。もっともすべての環境が整うためには 3 年程度(適合サイクル)を要すると考えられている。今のところ、半年毎に stable リリース、x.0、x.5 を出していき、Parrot 3.5(green fields)までの計画がある。その先は、Parrot を使う様々なアプリケーションの開発に主軸を移していくということらしい。ここまで来ると夢は限りない。プログラミングの基盤となるのだから、あらゆることに Parrot が使われるようになるはずだ。詳細は、次のページを参照すること。

The Vision for 1.0 | Parrot VM
http://www.parrot.org/news/vision-for-1_0

Rakudo Perl 6 は、Parrot 1.0 リリースの二日後、3 月 19 日に出る。現時点では Windows でも Parrot 0.9.1 に Rakudo の 20090220 版をセットアップして使うことができる。下記のサイトからダウンロードして使ってみよう。

Packages & Source Code | Parrot VM
http://www.parrot.org/download

(jscripter)

# やｎでレPython
# ～新約蛇神祭祀書～

*TIM: I warned you, but did you listen to me? Oh, no, you knew it all, didn't you? Oh, it's just a harmless little bunny, isn't it? Well, it's always the same. I always tell them--*
*ARTHUR: Oh, shut up!*

*ティム（妖術師）：あ～あ、だから言わんこっちゃねぇ。あんた、本当にわかってたのか？ 無害な小ウサギ？ いっつもそうだ。いつでも皆そう言う……*
*アーサー：黙れっ！*

*『MONTY PYTHON AND THE HOLY GRAIL』*

■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(victim)
・羽生一子（はぶ　いちこ）　　：パイソニスタ(Pythia)

## ２．バラ色ですか？　ブルーですか？

「ま、マナブくん」
放課後、帰り支度を始めたマナブを呼んだ、聞き馴れた声。
しかし、その声の調子もさることながら、今まで名前では呼ばれなかった筈。
振り向いて確認すると、間違いなくイチコだ。

**《名前呼びイベント発生》**

マナブの**ゲーム脳**に、テロップが浮かぶ。
そして選択肢も……。

**１．「何、羽生さん？」と、いつものように聞き返す。**
**２．「何、イチコさん？」と、さりげなくこちらも名前呼びする**
**３．「気安く名前で呼ぶんじゃねぇ」と意味なく切れる**

（３が論外なのはともかく、ここはやっぱ、アレっしょ）
「な、何？　イチコさん？」

**《選択肢２を選ぶ》**

……さりげなく答えたかったが、ちょっと噛んだ。
途端、イチコの顔がさぁっ、っと朱くなり、そして涙ぐむ。
（え？　え？　せ、選択肢間違えたかっ？）
あわてふためくマナブだが、イチコは涙ぐんだままにっこりと笑って「よかったぁ」と言った。
笑った拍子に頬を涙が伝って落ちた。
紛らわしい反応だが、どうやら正解らしい。

「ど、どうしたの？」
「え？　えへへ。ちょっと嬉しくて」
　朱くなりながら、笑いながら、ちろっと舌を出すイチコ。
　その仕草には、マナブでなくとも心臓に直撃するような可愛らしさがあった。
「ゆ、夢を見たんだよ」
　イチコがまた、脈絡のなさそうな話を始めるが、毎度のことで慣れたマナブは黙って聞く。
「今と同じように呼びかけたらね。夢の中のマナブくんは**『馴れ馴れしく呼びかけるんじゃねぇっ！ヘビ使いの売女がぁっ！』**ってののしるんだよ」
（イチコさん、僕をどんな風に見てるの？）
「だから……だから、**思わずナイフで刺しちゃった**の」
（！！）
「**何度も何度も刺して、返り血を浴びて真っ赤になって**、それでも夢の中のマナブくんはあたしをののしるんだよ。だから最後に、**斧で首を切り離しちゃった**んだ」
（……）
「でも、その首がいつまでも喋うんだよ。**斧で何度も何度も叩いて、叩き潰して、顔はなくなったのに**、声だけ響くの。耳をふさいでも聞こえるから、あたしも絶叫して……それで目が覚めたんだよ」
　壮絶な悪夢だ……マナブにとって。
「あ、あはは、夢の僕はひどいなぁ。それにイチコさんもひどいよ。僕がそんなこと、言うわけないじゃないか」
「そうだね、ごめんねマナブくん。多分ね、不安になってたんだと思う。今が幸せだから……」
　しおらしく謝るイチコを見て、その姿にグロテスクな恐怖を忘れかける。
　しかし、さっと後ろに隠した手に、なにやら光るものを見た気がした。
「……イチコさん、後ろに隠したの、それ、何？」
「え？　あ、あはははは」
　一生懸命ごまかそうとして、なにやら操作するが、本来的にドジなイチコのことである。
「あ！」
　ごとん、と音がして、ジャックナイフが床に落ちた。

「プログラミングは、実行方法に関して、大まかに２種類に分けられるんだよ。一つは**一括式(batch)プログラム**、もう一つは**対話式(interactive)プログラム**」
　前回と同じくマナブの部屋。
　既にPythonインタープリタを立ち上げた状態で、イチコの講習が始まる。
「一括式と対話式？」
「うん。一括式のプログラムは、必要な情報を渡したら、処理終了までノンストップのプログラム。画面では見えないところで動いているものが多いし、大半が一瞬で終わっちゃうから、気がつかないものも多いかもね」
　マナブに思い当たるものがあった。
「もしかして、立ち上げ時にやたら時間がかかるのも？」
「あはっ、そうそう。実はパソコンが『使える状態』になるように、大量の一括式プログラムが動いてるんだよ」
「やっぱり……でも、僕は素人だからわかんないけど、Windowsの立ち上げとか、もうちょっと省略できないのかな？」
「もしかしたらもっと楽に立ち上がる方法があるかもしれないけど、Windowsは複雑になりすぎちゃったから、整理が難しいかもね」
　なお、WindowsにせよMacにせよ、長いこと使い続けているパソコンは、どうしても後付けでインストールした常駐プログラムの関係で動作は若干遅くなる。
　ただし、その全てが本当に必要なプログラムなのかどうかは、不明だ。

「それにマナブくん、インターネットにつないでるから、ウィルス対策ソフト入れてるよね。これの立ち上げに時間かかってるってトコあるかもね」
　インターネットにつないでいないのなら、ウィルス対策は、新たに接続するストレージ（USBやCD/DVD、リムーバブルHDD、古いところではFDDなど)のチェックだけで済む。
　しかし昨今のOSは(Linux系も含め)、インターネットにつないであるネットワークコンピュータが想定されているので、ウィルス対策やスパイウェア監視ソフト、あるいは不正アクセス監視ソフトは必須だ。
「ウィルスは怖いもんね。まぁいいや、先を続けて」
「うん。対話式は、随時人間が必要な情報を打ち込みながら進めるプログラム。簡単なものではインストールウィザードみたいなものもあるけど、ワープロみたいな大きなプログラムも、対話式のプログラムの一種だよ」
「アプリケーションプログラムってやつ？」
「アプリケーションが必ずしも対話式とは限らないから、ちょっと違うかも。つまり、一度動かしたら止まるまで手出しできないのが一括式。……非常手段を使って強制終了しないかぎり、だけど。で、確認しながら進めるのが対話式だよ」
　マナブは少し考えて、ぽん、と手を叩いた。
「画像ダウンロードのための巡回ソフトって一括式かな？」
「うん、そういうのは一括式……って、**マナブくん、何の画像の話？**」
　ひぃっ、と、イチコの**『つめたいまなざし』**を浴びて硬直するマナブ。
「**まだ、画像集めてるんだ……**」
「に、**二次元**だけだから、ホントっ！」
「二次元？」
「人間の書いた絵だけだよ。三次元の人間を写した写真や映像は全部処分したっ！」
　イチコは、無言でPythonの立ち上がった画面に、なにやら打ち込む。

```
>>> print()
>>> print('♀'*10000)
♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀
♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀
♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀♀
……
```

　画面一面の『♀』の文字。
「**一次元だよ。楽しい？**」
　くすくす、と笑うイチコに、思わず背筋が寒くなる。
「あ……う……二次元もなるべく控えます」
　止めます、と言えないところが煮えきらないが、正直ではある。
「約束だよ？」
　イチコもそれで妥協したらしい。
　しばし流れる微妙な空気。
「い、一括式プログラムの話だったよね。今回はそれを書くの？」
　無理やり話を戻すマナブ。
「……そのつもりだったけど、マナブくんは対話式の方が好きみたいだから、そっちから始めるね」
「別に対話式の方が、なんて……」
「**一括プログラム、好き？**」
「え？　……あ、あの……」
「一括は問答無用だよ、**言い訳聞かないよ？**」
「……た、対話式って素晴らしいなぁ……」
　生きてるって、素晴らしい。

「一括式のほうが簡単なんだよ、一旦走ったら止まらないから。でも、言い換えると、止まる手をあらかじめ打って置かないと、止まれないよ」
「手っ取り早いけど、投げっぱなしってこと？」
「そうだね。対話式は、一々処理を中断して人間にお伺いを立てるから、面倒だけど安心、かな？」
「じゃぁ、さっきみたいに一度に文字を出すのは一括式？」
イチコは人差し指を口にあてて、ん〜、と考える。
「このインタープリタモード自体が対話式だから微妙だけど、あの１行のコードは一括だよ」
「それに、なんか文字列を掛け算とかしてなかった？」
「うん。文字列は、加算演算子（＋）と乗算演算子（＊）が使えるんだよ」
「減算（−）とか除算（／）は？」
「書いても、意味わかんないと思わない？」
「そだね」
「加算は文字列同士、乗算は文字列と整数にだけ使えるよ」

```
>>> 'spam' + 'egg'
'spamegg'
>>> 'spam' * 100
'spamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspam'
>>>
```

「すぱむ？」
「Python でのお約束だから、深く考えなくていいよ」
「スパムメールとかに、関係あるの？」
「オリジナルは同じみたい」
「まぁいいや。ところでコレって、カッコを使った計算とか出来るの？」
「出来るよ」

```
>>> print(('spam ' * 5 + 'and egg¥n')*10)
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg
spam spam spam spam spam and egg

>>>
```

「……あれ、『**￥ｎ**』って何？」
「前に言ったエスケープシーケンシーの一つで、改行記号だよ。『**￥**』って表示されてるけどこれは本当は日本語環境の文字化けで、ホントは『**＼(バックスラッシュ)**』なんだよ」
「ふーん、とりあえず改行を入れたければ、『￥ｎ』って書けばいいんだね」
「普通に改行を入れた文章を書くだけなら、トリプルクォートを使えばいいよ」
「トリプルクォート？」

```
>>> print('spam ¥nand egg')
spam
and egg
>>> print('''spam
... and egg''')
spam
and egg
>>>
```

「『’’’』や『”””』は、改行を含んだ文字を書けるんだよ。同じ『’』や『”』でも、三つ続かないと終止記号とはみなされないから、こんな書き方も出来るんだよ」

```
>>> print('''spam'spam'spam"and"egg''')
spam'spam'spam"and"egg
>>>
```

「シングルクォートとダブルクォートの混在した文字列？」
「そう。便利でしょ？」
　プログラミング経験が皆無なマナブには、今一どこが便利なのか実感がわかなかったが、とりあえず頷いておいた。
「えっと、対話式プログラムの話だったね。ごめんね、ちゃちゃ入れして脱線ばっかりさせて」
　イチコはふるふると首を振る。
「いいんだよ、マナブくんの興味の赴くままが一番。プログラミングは勉強じゃないんだから、どういう順番って、関係無いと思うよ」
　こういう『ただの教えたがりではない』ところは、マナブにも心地よかった。
　それだけに、イチコの期待に応えたいという気持ちも高まる。
「それじゃ、対話式プログラムの基本に入るね。まず、コンピュータのプログラムは、基本的には三つの制御構造の組み合わせで全て書けるって知っておいて」
「三つ？」
「うん、**構造化定理(structured theorem)**って言うんだよ。ところでマナブくん、ゲームブックって知ってる？」
「うん。昔流行ったらしいけど、今も復刻で少し出てるから。親父に奨められてちょっとやったけど、アナログで面白かったよ」
　イチコは微妙な顔で苦笑する。
「……マナブくんのお父さんって、色々侮れないね。ま、あたしとしては、好都合かな。色々説明省けるし。それで、ゲームブックって、選択肢によって、いろいろあちこちに飛ぶでしょ？」
「うん」
「実は初期のプログラムは、ゲームブックみたいに制御があちこちに飛んでたんだって」
「制御って？」
「どこの命令を実行するか、っていう順番のこと」
「でも、コンピュータだから、間違いなく実行するんでしょ？」
「コードに間違いがなければ、ね」

「あったら？」
「**大変**」
「……」
　なんか、ブラックホールのように重みのある『**大変**』の一言に、絶句するマナブ。
「大変、だったんだよ。だって、間違いがどこなのか一つ一つ辿っていかないとわからないのに、制御があちこち飛んでるから、プログラムを辿ること自体が大変だったんだよ」
「えっと……たしか親父にソレ、聞いたことあるぞ。スパゲッティコード、とかいうの、それじゃない？」
「……マナブくん、時々スゴいね。マナブくんのお父さん、怖るべし」
「ははは、聞き齧りだよ」
　心中でこっそり父親に感謝するマナブ。
「そう、正にスパゲッティ状態にからんで、わかんなくなっちゃうんだ。だから、プログラムの書き方に一定のルールを決めたんだよ」
「……それって、書けないプログラムがでてくるんじゃないの？」
「うん、そう思うのは当然なんだけど、実際は**その三つの構造だけで、いかなるコードでも書ける**ことが、**数学的に証明されて**いるんだよ」
「うぁ、す、数学的？」
　数学的即ち論理的。
　数学的に証明されたことは、必然的絶対。
「うん、その証明の内容はあたしも知らないけど、全てが記述できるって保証だけで十分。そうじゃない？」
「ま、まぁ、考えてみればそうだよね」
「それが、構造化定理の三構造。**順次(sequence)、選択(selection)、反復(iteration)**の三つだよ」
「うん」
「順次は簡単、単に上から下に順番に処理されるってこと」
「……それ、構造の一つに入るの？」
「うん。だから制御構造らしい構造って、本当は２つなんだよね。ここからは、実際にコードを動かしながら、見てみようか。まずは選択、の例」

```
>>> x = 1
>>> if x == 1:
...   print('Yes!')
...
Yes!
>>>
```

「……」
「わかった？」
　マナブは無言で首を振る。
　イチコは苦笑しながら、ディスプレイの文字を指して言った。
「じゃ、ワンステップづつ説明するね。まず、**if**だけど、英語の意味は知ってるよね？」
「『**もし～ならば**』だっけ？」
「そう。『～ならば』の部分は、ifの後につながる部分だよ。今回は『ｘが１ならば』ってなるんだよ」
「エックス、イコールイコール？」
　等号が二つつながっている記号に、マナブは当然奇異を覚える。
「うん。イコール一回は代入だから、二回連続でフツーの等号の意味になるんだよ」
　理解はできるが、納得はできない。
「代入の方を別の記号使った方が、わかりやすくない？」

「あはっ、そうかもね。でも、代入を書く場所って、同じかどうかを調べる同一性テストを記述するより頻繁に出てくるんだよ。だからよく使う方を簡略化して書けるようにしたみたいだね。勿論プログラミング言語には色々あって、Pascalとか Smallalk なんかでは、代入に『：＝』を使うから、マナブくんみたいに考える人がいないわけじゃないよ。でも、一番メジャーなＣ言語の方式に倣(なら)ってる言語が今は多いかな」
「Python も、そのＣ言語式？」
「そうだよ。昔は一部、Pascal っぽい記述も使えたんだけど、今は廃止されてるよ」
「ふ〜ん、要するに慣例みたいなものなんだ」
「そうだね。例えばさっき名前を出した Smalltalk では、本式には『←』を使うんだけど、『←』の記号が、昔よく使われた文字コード表に無かった関係で、『：＝』で代用されるようになったんだよ。だからもし、その文字コード表……ASCII に『←』が入ってたら、代入が『←』になって、同一性テストは『＝』になってたかもね」
　最新のテクノロジーの代名詞のようなコンピュータが使う言語に歴史っぽい一面があることに、マナブは妙に感心する。
「続けるね。『**if x == 1**』で、『**もし、ｘが１ならば**』で、その後に**：(セミコロン)**を打つんだよ。これは**区切り文字(delimiter)**といって、条件節がそこで区切られることを意味するんだよ」
「条件節？」
「命令文の中で、以降の命令を行う『条件』を記述する部分だよ。今回の場合**ｘが１でなければ、以降の命令は行わない**、という意味なんだよ」
「以降の命令を行わない……って、そこでプログラムはおしまいって意味？」
　イチコはペロリと小舌を出す。
「あ、ごめんごめん。あたしの説明が間違ってた。正確には『**以降の一連の命令**』だよ。一連の命令ってのは、字下げ(インデント)されてそろってる部分。『**ブロック**』っていうんだけど、その部分だけ、条件に合わなければスキップするっていうイメージかな？　さっきの例、もう一回、ちょっと変えてやってみるよ」

```
>>> x = 0
>>> if x == 1:
...     print('Yes!')
...
>>>
```

「'Yes!'が出なかった？」
「そーいうこと。今はブロックに一つだけど、２つ以上書いても同じだよ」

```
>>> x = 0
>>> if x == 1:
...     print('Yes!')
...     print('Yes2!')
...
>>>
```

「今度は、条件に合うように書いてみるね」

```
>>> x = 1
>>> if x == 1:
...     print('Yes!')
...     print('Yes2!')
...
```

```
Yes!
Yes2!
>>>
```

「わかった？」
「……」
「わかんない？」
「ううん、動く理屈はわかったけど……」
「けど？」
「何に使うのかわかんない」
　イチコは苦笑した。
「そうだね。確かにこのままだと、使い方わかんないよね。それじゃ、一旦インタープリタを閉じて、テキストエディタを開いてみて？」
「テキストエディタ？　メモ帳でいい？」
「うん、はじめはそれで十分。慣れてきたらプログラム記述専用のテキストエディタを用意するといいよ」
「じゃぁ」
　メニューからメモ帳を開くマナブ。
「これから言う通りに打ち込んでみてね」

```
x = input('一桁の数字を入力：')
if x == '3':
  print('正解です')

input()
```

「これに、test01.pyって名前をつけて保存してね。あ、そうそう、**文字コードはUTF-8**を指定してね」
「UTF-8？」
「うん。全角文字が混じってなければそのまま(ANSI)でもいいんだけど、全角文字を含める場合はUTF-8を指定しないとエラーを起こしちゃうんだよ」
「えっと……普通のシフトJISとか使えないの？」
「使えるけど、スクリプトの最初で文字コード指定しないといけないから、面倒なんだよ。UTF-8なら普通に通るから、簡単だよ」
「じゃ、UTF-8で」
「セーブ場所は、とりあえずデスクトップ上にしておこうか？」
「了解。……にゃ？　なんか見覚えのあるアイコンが出た」
　デスクトップに現れたのは、黄色と青の蛇が絡まる形が、昔なつかしプリントアウト用紙の右下方についているアイコン。
「Pythonスクリプトのアイコンだよ。WindowsにPythonがインストールしてあると、拡張子がpyのテキストは、このアイコンで表示されるんだよ。そしてダブルクリックすると、プログラムが実行されるんだよ」
「うわぁ、プログラムみたい」
「**みたい、じゃなくってプログラムだよ**」
「え？　だって、ただのテキストだし。えっと……そうそう、たしか『コンパイル』とかしてないし……」
　イチコは難しい顔をした。
「**プログラムに必ずしもコンパイルは必要じゃないよ。**機械が解釈できる形式に忠実に記述したアセンブリ言語は、アセンブル(assemble 組立)してメモリ上に乗せれば実行できるよ。それに、さっきのPythonインタープリタは、一行ずつそのままインタープリット(interpret 解釈／翻訳)しながら実行

してるから、コンパイル(compile 編集)はしてないよ」
「……とりあえずダブルクリックしてみるよ」
　知ったかぶりで、いい加減なことを言うのではなかった、と後悔しながらアイコンをダブルクリック。
　しばらく何か動いた後、コマンドプロンプト画面が開く。

```
一桁の数字を入力：
```

「とりあえず３を入力してみるか」

```
一桁の数字を入力：3
正解です
```

「……」
　エンターキーを押すと、プロンプト画面は消えた。
　もう一度、リトライ。

```
一桁の数字を入力：
```

「３以外の数値を……」

```
一桁の数字を入力：5
```

　プロンプト画面は無反応。
　もう一度エンターキーを押すと、やはりプロンプト画面は消えた。
「……」
　情けない顔でイチコを見るマナブ。
「えっと……面白くない？」
　こくこく、っと頷くマナブ。
「まぁ、初めてのスクリプトだと、あんまり気の利いたもの、作れないものねぇ」
「せめて、間違った時のメッセージって出ないの？」
「ああ、それなら簡単だよ」
　メモ帳を開き、そのウィンドウにスクリプトファイルを落とす。

```
x = input('一桁の数字を入力：')
if x == '3':
  print('正解です')
else:
  print('不正解です')

input()
```

「**else**？」
「意味、知ってるよね？」
「えっと……『**そうでなければ**』だっけ？」
「そう。if の後のブロックは、条件に合った時に実行される文。else の後のブロックは、条件に合わなかった時の文」
「なるほど合点。ところで、さっきから気になってるんだけど、input って何？」
「input は、**標準入力**っていわれるところからの入力を求めて、その値を返す関数だよ。標準入力は、

とりあえず何もしなければ、コマンドプロンプト画面からのキー入力につながってるんだよ」
「３がシングルクォートで囲まれているのは、なぜ？」
「**input 関数**はキー入力を『**文字**』として受け取るんだよ。インタープリタに直接入力する場合は数字として受け取ってたけどね」
「文字として受け取った数字は、数値にならないの？」
「なるよ。**eval 関数**を使えばね」

```
>>> x0 = input()
10
>>> x = eval(x0)
>>> x0
'10'
>>> x
10
```

「eval 関数は、本当はもう少し難しい動作をする関数なんだけど、とりあえず数字を数値に変えることはできるよ」
「あと、もう一つ。最後の input 関数って何？」
「抜いてみればわかるよ」
　言われた通り、抜いてみた。
　その結果……
「あ、あれ？　結果見えないうちに消えた？」
「そういうこと。コマンドプロンプトを立ち上げて、そこから実行するなら、最後の input は要らないんだけど、スクリプトファイルを叩いて実行する場合、input で入力待ち状態にしておかないと、表示されてすぐにプログラムが終了して、**ウィンドウが閉じちゃうんだよ**」
「なるほどね」
「というわけで、条件分岐は解った？」
「なんとか……多分。ところで、分岐の中で分岐する時には、字下げしてブロックの中で分岐するんだよね？」
「そうだよ」
「じゃ、文字を数値で受け取る方法を含めて、これで動くかな？」

```
x0 = input('一桁の数字を入力：')
x = eval(x0)
if x == 1:
  print('10点です')
else:
  if x == 2:
    print('20点です')
  else:
    if x == 3:
      print('30点です')
    else:
      print('0点です')

input()
```

「論より RUN、と、ダブルクリックっ！」

```
一桁の数字を入力：1
10点です

一桁の数字を入力：2
20点です

一桁の数字を入力：3
30点です

一桁の数字を入力：4
40点です
```

「やった☆　予想通り」
　うれしそうに笑うマナブを見て、イチコも微笑む。
「マナブくん、楽しい？」
「うん。なんか予想して、予想通り動くって楽しいね」
「それが**プログラミングの楽しさ**だよ。いろいろ覚えていくと、できることが増えていく、そして予想通り動くと楽しい。今、マナブくんは、あたしが何も教えていないことを予想してやったよね？それって凄いことだよ」
「えへへ。でも、なんかちょっとまどろっこしいなぁ。もうちょっと簡単に書ける方法はあるのかな？」
「あるよ」

```
x = eval(input('一桁の数字を入力：'))
if x == 1:
  print('10点です')
elif: x == 2:
  print('20点です')
elif x == 3:
  print('30点です')
else:
  print('0点です')

input()
```

「えるいふ？」
「意味的には**else-ifの省略形**だけど、elseと違って、ブロックを入れ子にせずに、**複数分岐**できる構文だよ」
「あと、evalのかっこの中にinputが？」
「代入した式をそのまま書いただけ。関数は式だから、変数をそのまま式に置き換えても問題無いんだよ。難しいように見えるけど、原理は簡単なんだよ」
「……やっぱ、そーいうスマートな書き方あるんだよなぁ……」
　少し落ち込むマナブの手を取って、イチコは大きくかぶりを振った。
「ううん、プログラムは、自分の知識の組み合わせで作るものだから、マナブくんの答えは正しいよ。同じ結果を出すだけなら、もっと他にも色々方法はあるけど、それはおいおい覚えていけばいいことだよ。**プログラムは覚えるんじゃない、組むんだよ。**いろんなことができるけど、結局自分のやりたいことは、自分で解法……アルゴリズムをつくっていくしかない。だからプログラミングは、創造的な作業なんだよ。決して覚えたことをそのまま打ち込むものじゃないんだよ」
　マナブは言葉の内容より、一生懸命しゃべるイチコの姿に見とれていた。

じっと正面から見つめるマナブの視線に気づくと、イチコは恥ずかしそうに顔を赤らめた。
「あ、ちょ、ちょっとヒートアップしちゃったかな」
「え？　う、うん」
「でも、ホントはこのコード、不完全なんだよ」
「え？」
「キーボードから打ち込めるのは数字だけじゃないんだよ。もし、数字以外の文字を打ち込んじゃったら……」
と言いつつ、そのスクリプトを実際に動かしてみる。

一桁の数字を入力：a

「何か一瞬見えたけど……」
「メッセージが読めずに終了しちゃったよね。文字を打ったら必ずエラーするわけではないのだけど、最悪、プログラムが強制終了されちゃうことがあるんだよ」
「そっか、予想してたのと違う場合って、考えてなかったなぁ」
「自分で作って自分で使うプログラムならいいけど、入力を伴う対話式のプログラムの場合、出来る限りの場合は想定しておかないとトラブルの元なんだよ」
「どうすればいいの？」
「本当は文字のまま判定した方が安全なんだよ。もし数字に直すなら、数字にならない場合の対策をとるべきなんだよ」
「対策？」
「**例外処理**っていう部分を入れるんだよ」

```
try:
  x = eval(input('一桁の数字を入力：'))
except:
  x = 0

if x == 1:
  print('10点です')
elif: x == 2:
  print('20点です')
elif x == 3:
  print('30点です')
else:
  print('0点です')

input()
```

「とらい？　えくせぷと？」
「**try** はその通り『**やってみる**』こと。**except** は『**～を除いて**』の意味で、具体的にはエラーが出た時の処理を書くんだよ。except の後ろには具体的にエラー名を書いてもいいんだけど、どんなエラーでもとりあえずスルーしてしまうなら、エラー名を書かないこともあるんだよ」
「エラー名を書くとどうなるの？」
「例えば、数値以外の文字に対するエラーなら **NameError** だから、

```
except NameError:
```

と書いてもいいんだけど」

「けど？」
「エラーを限定すると、それ以外のエラーを拾ってくれなくなるんだよ。たとえば、NameErrorを指定したバージョンで……」

```
一桁の数字を入力：a
0点です
```

「……は、大丈夫だけど……」

```
一桁の数字を入力：5/0
```

「あ、消えた」
「うん、これは書式としては正しいんだけど、５を０で割ると答えは出ないよね。だから**ZeroDivisionError**っていう別のエラーが出たんだけど、NameErrorに限定してるから、エラーを拾いきれなかったんだよ」
「じゃぁ、エラーは限定しない方がいいんだね？」
「そうとも限らないよ。プログラムに関わる深刻なエラーの場合もあるから、エラーは全部無視すればいいとは思わない。今回は無視してもいいけど、一応どんなエラーがあったか、表示する方法はあるんだよ」

```
try:
  x = eval(input('一桁の数字を入力：'))
except Exception as E:
  print(E)
  x = 0

if x == 1:
  print('10点です')
elif: x == 2:
  print('20点です')
elif x == 3:
  print('30点です')
else:
  print('0点です')

input()
```

「**Exception**は、全てのエラーに対応するけど、その後にasを書いて変数を指定すれば、変数……今回はEだけど……の中に、エラー内容を入れてくれる。これを表示すれば、なにが間違いだったか、一応、わかるよ」

```
一桁の数値を入力5/0
int division or modulo by zero
0点です

一桁の数値を入力aiu
name 'aiu' is not defined
0点です
```

```
一桁の数値を入力 1++
unexpected EOF while parsing (<string>, line 1)
0 点です
```

「……英語だね」
　不満そうなマナブの顔に、苦笑するイチコ。
「仕方がないよ。一応説明すると、最初の は 0 で数値を割った ZeroDivisionError、次のが『そんなオブジェクト無いよ』の意味の NameError、最後が『**文法が違う**』の **SyntaxError** だよ」
「でも、人に渡しても、正直わかんないと思うけど？」
「そうだね。もしマナブくんが人に渡すものを作るのなら、予想できるかぎりのものについては、メッセージを独自につくっておくといいかもね」

```
try:
  x = eval(input('一桁の数値を入力：'))
except ZeroDivisionError:
  print("数値をゼロで割りました")
  x = 0
except NameError:
  print("オブジェクトに無い名前を入力しました")
  x = 0
except SyntaxError:
  print("文法上の誤りがあります")
  x = 0
except Exception as E:
  print("何かエラーがありました")
  print(E)
  x = 0

if x == 1:
  print('10 点です')
elif x == 2:
  print('20 点です')
elif x == 3:
  print('30 点です')
else:
  print('0 点です')

input()
```

「**except は複数書けるの？**」
「**いくつでも必要なだけ書けるよ**」
「Exception って、全部のエラーを受け取るんだよね。そうすると、例えば ZeroDivisionError で引っかかった後、**もう一度 Exception で引っかからないの？**」
「最初に引っかかった except 節のブロックを実行した後は、**それ以降の except 節は無視する**んだよ」
「便利だね」
「例外処理には色々な書き方があるけど、今回はコレくらいにしておくね」
「詰め込みすぎは良くない、だね」
「そうそう」
「お茶淹れるよ」

マナブが紅茶のポットを持って現れると、案の定、イチコがかちかちとマナブのパソコンをいじっていた。
「別に変なもの、入ってないでしょ？」
マナブは苦笑しながらローテーブルに紅茶を置く。
イチコは、その横に添えられたシュークリームを見つけ、目を輝かせる。
「わ、それもしかして『キラーラビッツ』の？」
「うん、おいしいって聞いたから」
「ふふふ、ありがと」
甘いモノを無邪気に頬張るイチコを見ながら、マナブは改めて自分の選択に間違いがなかったことを確認する。
（いいじゃないか、少しくらいヤキモチ焼きだって。こんなに可愛いんだから）
しかし次の日、目覚めのメールチェックのためにＰＣを立ち上げたマナブは、凍りつくことになる。

***――――おはよう、マナブくん――――***

イチコの肉声の立ち上げ音とともに、背景画として、証明写真大のイチコの顔写真がぎっしりと並んで、マナブを見つめていた。

**【内容】**
**第一回目：**
**・四則演算**
**・変数、代入（代入子＝）**
**・print関数**
**・文字列**
**・文字列のインデックス表示、スライス**
**・len関数**
**・exit関数、quit関数**

**第二回目：**
**・文字列の結合（＋）、反復（＊）**
**・『¥n』改行コード**
**・トリプルクォート表記**
**・if構文(if-else, if-elif-else)**
**・同一判定演算子（＝＝）**
**・input関数**
**・eval関数**
**・例外処理：try-except文**
**・例外：Exception, ZeroDivisionError, NameError, SyntaxError:**

**【注釈】**
※やや上級者向けの内容も含まれているので、初心者の方は、分らないところは読み流してください（分るところは、笑ってください）

**・前回の話題で分数モジュールのこと**
Python 3.0で追加されたfractionsモジュールのFractionクラスは、分数（有理数）が扱える。しかしやはり、複素数をコアで扱えるなら、分数もコアで扱えて欲しかった

**・（最初のコント）**
今回は有名な、『ホーリィグレイル』のキラーラビットのシーン

**・victim, Pythia**
『犠牲者』と『邪教の巫女』の意味

**・バラ色ですか？　ブルーですか？**
これ、ある歌の歌詞なのだけど、多分なんの歌かわかっても、誰にも褒めてもらえない**（寧ろキモがられる）**

**・名前呼びイベント**
イベントではなく、好感度のバロメータとして扱われる場合が多い

**・ゲーム脳**
間違っても、全ての現実（リアル）をゲームになぞらえる脳のことではない。ちなみに、『ゲーム脳』という本を読むと、実はRPGなら頭に良い、と書いてあったりする

**・ヘビ使いの売女**

ちなみにニシキヘビ(Python)は**男性のアレ**の象徴でもあるので……

**・顔はなくなったのに、声だけ響く**
一般的には、それを**幻聴**という
**・ジャックナイフ**
折りたたみ式の大型のナイフ。でも普通はバイオレンスジャックのアレほど大きく無い

**・一括式プログラム**
人間と対話しないプログラムであって、機械と対話していないわけではない

**・対話式プログラム**
命令される気分にはなっても、決して対話している気分になれないのはどうしてだろうか

**・画像ダウンロードのための巡回ソフト**
別にそれ自体が怪しいものでは決して無いはずだが、怪しい目的以外には滅多に使われないソフト

**・二次元**
コミックやアニメやＣＧなどの仮想世界を指す言葉として使われるらしい

**・♀**
女性、雌を現すマークだが、本来は天文学での金星（ヴェヌス）を表すマーク。火星（マルス）の♂のマークが雄のマークとして選ばれているのだが、要するにヴェヌスとマルスからクピト（愛）が生まれるという意味らしいが……つーと**愛とゆーのは全て不倫なんかい**、とツッコミを入れたくなる（反論はしない）

**・spam**
ご存知、Monty Python's Flying Circus の通称『スパムの多い料理店』スケッチより。うっとうしい迷惑メールをスパムメールというが、それはこのスケッチのバイキングの『スパム』の大合唱から来ているらしい。Python ではそれを繰り返しの例としてよく取り上げる。そもそもスパムとは、そーいう名前のハム缶のようなモノで、フレッシュバーガーに行くと『スパムバーガー』というのが食べられる。私は結構好き

**・spam and egg**
ちなみに『spam』を『スパーム』と長読みすると『sperm』に聞こえる。『egg』は『たまご』の意味も『らん』の意味も。そういう下品なネタなのだけどね、実際

**・バックスラッシュ**
Mac を使ってると、たまにそのまま表示されるのでびっくりする

**・トリプルクォート**
薀蓄は散々前回書いたのでパス。でも、トリプルクォートを使って、シングルとダブルの両方のクォートをエスケープする技は、私は他では見たことがない（できるかどうか心配だったが、試したら大丈夫だった）

**・構造化定理**
知ってる人は『またか』と思うだろうけど、やっぱり基本。基本３構造を守り、ランダムジャンプの廃止。Python は徹底的にランダムジャンプが書けないので、**そもそも構造化していないプログラムは裏技を使わないと書けない**

**・ゲームブック**
日本でも90年代に密かにブームがあった。最近、復活しようと復刻版が数冊出たが、消えてしまったらしい。当然だろう。コンピュータが手元にゴロゴロしてる時代に、番号を追ってなぞいられない。ケータイとかのソフトで出せば、まだ売れるかも。
ちなみにゲームブックを再現するソフトをPythonで組もうとすると、ランダムジャンプが出来ないことが仇になって難しくなりそうだが、ループの中で条件分岐すればさほどの面倒もなく再現できる

**・「あったら？」「大変」**
大変なんです

**・スパゲッティコード**
あるいは『スパゲッティプログラム』。BASICでプログラムを始めるとコレになることが多い。昔はそれを矯正させるためにPascalとかを書いたものだけど、今ならPythonで十分

**・数学的**
論理矛盾が無い、という意味。ただし、論理の根拠自体をその論理で説明することは出来ない……というのが不完全性定理というものらしい

**・Pascalっぽい記述**
以前は「同一でない」の同一否定演算子として『＜＞』が使えたが、Python 3.0で廃止され『！＝』に統一された。以前Pascalを使っていた筆者は『＜＞』派だったので少し悲しかった

**・区切り文字(delimiter)**
デリミタとも。Pythonの記述要素の一つ。定義文の宣言節や選択文の条件節の区切りなどに用いる

**・字下げ**
インデント(indent)歯を引っ込める、で窪地／引っ込みの意味から、レイアウトで先頭を下げて始める書式の意味。インデントブロックという、ある意味Pythonの嫌われ属性の一つだが、Guidoの主張するように、どうせ強要されなくてもインデントフォーマットするのだから、インデントを強要されるのではなく、C言語のブレース {} が省略された書式と考えてるべき。ちなみにPythonでも、一行程度で書ける文であればブロックを作らずにデリミタの後に続けて書くことができる

**・テキストエディタ**
パソコンユーザ＝プログラマの時代でなくなって、説明しにくくなったソフトの代表格。テキストエディタとワープロの違いを説明するには、テキストファイルのデータ構造に言及しなければならない（特にプログラム／スクリプトを書く場合には、単純に『簡易ワープロ』とごまかせない）のが面倒

**・文字コードはUTF-8を**
正確には、UTF-8は文字コードではなくエンコーディングだが、メモ帳(notepad.exe)の保存ダイアログにはそう書いてある。UTF-8(UCS Transfer Format 8)はUnicodeやUCS(Universal multiple-octet coded Character Set = ISO10646-1)の８ビットエンコーディングの一つ。本来16ビットで表現されるUnicode文字コードを、プログラム界で未だによく使われるASCIIコードの上位互換として使えるようにしたもの。文字コードとエンコーディングの違いは、はっきり言えば文字回りに関係するコアレベルなプログラムを書かなければあまり関係ないが、一口にUnicodeといっても色々あることは知っておかないと色々不便。とりあえずPythonで文字を使うならUTF-8と覚えておけば問題ない。ただし、プログラムの中に書かないのであれば、Pythonでは様々な文字コード／エンコーディングの文字が使えるようにcodecsというモジュールがセットされている

**・コンパイル**

実は、Python スクリプトを実行する際には、ヴァーチャルマシン（仮想機械）にかけるために、自動的にバイトコード（P コード = pseudo-code）にコンパイルされる。これは Perl などと同じ方法で、実行が終わるとバイトコードは消去される。実際のＣＵＰ用のマシンコードでなくヴァーチャルマシン用のコードを生成する方法は、古くは Pascal などで、現役なら Java が有名。ちなみに Python ではモジュールをインポートするとバイトコードのファイル（つまりコンパイルしたファイル）が残る

**・論より RUN**

BASIC 時代の合言葉で、『うだうだ考えていないで、動くかどうかは実行して試してみろ』の意味。暴走しても止められる BASIC 環境だから言える言葉であって、環境によっては暴走すると手が付けられない場合もあるので注意。ただし、Python 環境ではとりあえず（ほぼ）大丈夫

**・elif**

Python では、Ｃ言語の switch 文に相当する文法を if 構文の拡張として実現している。なお、Python の辞書(dictionary)を使うと、こんな書き方も出来なくは無い……

```
x = eval(input('一桁の数字を入力：'))

if x in (1,2,3):
  {1:lambda: print('10 点です'),
   2:lambda: print('20 点です'),
   3:lambda: print('30 点です')}[x]()
else:
  print('0 点です')

input()
```

このコードには（lambda の使用方法を含めて）様々な問題があるが、Python のコレクションの機能と、関数をファーストクラスオブジェクトとして簡単に扱える利点は、頭を柔軟に利用すれば様々な形に応用できる。

**・eval のかっこの中に input が**

戻り値を伴う（正確には戻り値が None でない）関数は、その戻り値と入れ替えられることを前提としてそこに式を書くことができる。ただし、評価の順番は必ずしも前から行われるとは限らないので、順序が影響するような場合は注意が必要（大方、前から評価されるようだが……）

**・覚えるんじゃない、組む**

実質上、プログラミングできるようになれるかどうかの境目。沢山の構文を覚えるより、自分の手持ちの構文をどれだけ使いまわせるかが大事。「正しい作法」より、まず「自分で考える」ことができなければならない。オブジェクト指向プログラミングだの、関数プログラミングだのは、自分流でとりあえずぶん回せるようになったら、で十分。実際のところ、今回紹介した if 構文と、次回以降紹介する反復の for 構文、while 構文、それに関数呼び出しが分れば、殆どのプログラムは書けてしまう。一通り書けるようになったところで、様々なスタイルを勉強してみるのは楽しくて為になるし、その必然性や威力も理解できるが、書けないうちに頭から色々なスタイルを勉強しても意味はほとんど無いと思われる

**・例外処理(exceptions)**

Python で多用される EAFP(Easier to ask for forgiveness than permission ゴメンナサイはオネ

ガイシマスより楽)スタイルの要。EAFP は『論より RUN』とはちょっと異なるが、とりあえず動かしてみてエラーが出ればそれに対処する、というスタイルで、先にがちがちに枷をはめてしまうＣ言語や Java 言語のようなスタイル(LBYL=look before you leap 飛ぶ前によく見ろ)の対極。コアなシステム言語では、勿論後者のスタイルが必須なのだが、その強固なシステムの上で**悠々と**コードを走らせることができる**優雅な** Python などの環境では、前者で十分ということ（無論、**その下に強固なシステムが走っていることが前提**となる）本質的には、発生する例外をつかまえて処理する if 文と同じ構造なのだが、大概の入門書では後の方で説明される。しかし、ファイルは標準入力など、入力内容の予想がつきにくい場合には、書いておく癖をつけると良い。しかし、**自分専用のスクリプトでは、当然だが滅多に書かない**

**・キラーラビッツ**

菓子屋の名前らしい。私の書くラノベ（本文のような入門テキストを含む）には、食べ物屋だの喫茶店がやたら出てくるが、その店の名前は大体何かをもじってある。

**・（最後のイタズラが Python スクリプトで書けるか？）**

書けるかもしれませんが、私なら素直に Windows の設定をイジります

***【Python あれこれ】１．関数(function)という呼称***

Python では、呼び出し可能であり、クラス(あるいはタイプ)やクラスインスタンスでないオブジェクトを『**関数(function)**』と総称しています。ただし、この『関数』という言葉は、実際にはあまり適切ではありません。数学でいう関数とは、「**あるＸに対してある値Ｙが決まる時、ＹはＸの関数である**」という意味なので、input 関数のように、**Ｘに当たる引数部分が決まっても、返す値が確定しないもの**に対して『関数』という名称は本来ふさわしくありません。最近流行った「関数プログラミング」でいうところの関数は、この数学的な意味での関数です。このような誤解を招かないために、関数を含めてプログラミングの一連の手順を表す言葉として『**手続き(procedure)**』という言葉があります。Scheme などでは、『関数』の代わりにこの『手続き』という言葉が使用されていますが、Python で『関数』といわれているものは総じて『手続き』と考えても問題ありません。逆に、例えば（今回は欄外でのみ登場した）辞書など、インデックスで指定できるコレクションオブジェクトは、本当の意味での『関数』だったりします（正確には、辞書については関数より範囲の広い『**写像**』ですが、Python で言う『関数』は写像ですらありません）Python は教育用にふさわしい言語だと思いますが、こうしたささやかな用語などにもう少し配慮してもらえると、もっと教育用として使いやすくなるかな、と思います。

*（機械伯爵）*

# 状態

Yさ

1. このゲームは

前作(KEY=いわゆる数列を高度な判断で推理するゲーム)に引き続き、貧相な画面のゲーム第五弾(^^;)

今回は、A～Zの文字を使用した文字列を高度な判断で推理するゲームです。

(...って結局数当てと似たようなものなのですが(^_^;)

前作KEYの「短い桁のが欲しい」とのリクエストを元に、数字を英字にして、答えが入力によって変化するといったひねりを加えてみました。

...要するにスクリプトを比べるとほぼ同じ構造、という事ですd(^^;

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)行なってます。

3. 遊び方は

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

等で起動するとゲームスタートです。

1)A～Zの文字を使用した文字列をレベル数分の文字数で入力して下さい。

A～ZはZの後ろがA(Aの前がZ)として連続しているとし、ヒントが次のように表示されます。

'+' : 入力した文字が 答えより後方にある場合
'-' : 入力した文字が 答えより前方にある場合
'*' : 入力した文字が 答えと前後同位置の場合
'@' : 入力した文字が 答えと一致している場合

```
ヒント    * -       ←         - @ +       →         +
答えがN : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
答えがZ : M N O P Q R S T U V W X Y Z A B C D E F G H I J K L
```

なおヒントが'+'か'-'の場合は答えがその方向に1つずれます。

例えば答えが Z で'+'の入力だった場合は答えが A に変わります。

2)全桁を当てるとラウンドクリアですが、8回以内に当てないとゲーム終了となります。

3)ラウンドクリア毎に問題のレベル=文字数が1つ増えます。

最終ラウンドまで当て続けて下さい。

ちなみにスクリプトの LAST_ROUND の値で最終ラウンドを変更できます。デフォルトは5となっています。

※なお ROUND_LIMIT は目安ですので、気にしなければ10より大きくできます
d(^^;

## 4. 開発環境など

ソースは、例によってかなり昔に会社の行き帰りの満員電車で立ったまま作成したC言語版を、今回awkに移植したものです。

相変わらずマシンは HP100LX でした v(^^;) が、HP も 200LX の生産中止にしちゃったし、液晶パネル蓋の止め具はバカになっているし(プリペイドカード再利用でちょっち復活)、電池入れ蓋はセロテープで止められているし、液晶パネルのヒンジが突然もげて瞬間接着剤でかろうじて固定されていたりするし、(地球にやさしくないなと思いつつも)アルカリ単三を2本づつ消費し続けてくれていて、他のマシンには絶対到達できない便利さ&タフさなのでした。

しみじみ。

## 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。

(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。

[STATUS.awk]

```
##  状態  written by Ｙさ

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

BEGIN{
##
## 定数定義
##
LAST_ROUND =5
# ROUND_LIMIT =10  ## 設定可能な最終ラウンド（目安）

  last_round=LAST_ROUND;  # default


# 答え格納用
# ans[10], org[10];

# 文字->数値化用
  split("ABCDEFGHIJKLMNOPQRSTUVWXYZ", alTbl, "");
  for(i=1; i<=26; ++i) nuTbl[alTbl[i]]=i;
```

```
  # ゲームメイン
  doGame();
}


##  HELP 表示
function helpDisp(){
  print "";
  print "  レベル数分の文字数の A～Z の文字を使用した文字列を当ててください";
  print "";
}


## 解答チェック
function ansChk(level, s,  i,r){
  r=0;
  for(i=0; i<level; ++i) if(alTbl[ans[i]]==toupper(substr(s, i+1, 1))) ++r;
  return r;
}


## 答え(ヒント)表示
function ansPut(level, s,  i,sw,num){
  printf("        ");
  for(i=0; i<level; ++i){
    if(s==""){ printf(alTbl[ans[i]]); continue; }
    sw=chkSub(ans[i], nuTbl[toupper(substr(s, i+1, 1))]);
    if(sw==13 || sw==-13) printf("*");
    else if(sw<0)  printf("-");
    else if(sw>0)  printf("+");
    else           printf("@");
    num=ans[i];
    if(-13<sw && sw<0) --num;
    if( 13>sw && sw>0) ++num;
    if(num<nuTbl["A"]) num=nuTbl["Z"];
    if(num>nuTbl["Z"]) num=nuTbl["A"];
    ans[i]=num;
  }
  if(s!="") print "";
  else{
    printf(" {"); for(i=0; i<level; ++i) printf(alTbl[org[i]]); printf("} ");
  }
}
function chkSub(base, test,  dif){
  dif=test-base;
  if(dif < -13) dif+=26;
```

```
  if( 13 < dif) dif-=26;
  return dif;
}


## 答え設定
function ansSet(level,  i){
  for(i=0; i<level; ++i) org[i]=ans[i]=nuTbl["A"]+rnd(26);
}


## 入力チェック
function notAll(level,s,  i,l,p)
{
  # help ?
  if(s!="") p=toupper(substr(s, 1, 1)); else p="?";
  if(p=="?"){  #  help !
    helpDisp();
    return -1;
  }

  l=0;
  for(i=1; i<=length(s); ++i)
    if(toupper(substr(s, i, 1)) in nuTbl) ++l;
  if(l!=level) return -1
  return 0;
}


##
##  ゲームメインループ
##
function doGame(  v,c,s,h,b,round,level,t){
  srand();

  h=0;
  for(round=1; round<=last_round; ++round){
    s=0;  ansSet((level=(round<5)?round:5));
    printf("¥n*** Round%02d (Level%02d) ***¥n¥n",round,level);
    for(c=1; c<=8; ++c){
      do{ printf("[%2d] >> ", c);  t="";  getline t; }while(notAll(level,t));
      if((v=ansChk(level,t))==level) break; else{ ansPut(level,t); s+=v; }
    }
    ansPut(level,"");
    printf("  %s¥n", ((v==level)?("Round clear"):("Over times")));
    print "";
    if(s>0)      printf("          Hit point %02d¥n", s);
```

```
    b=(round<5)?1:3;
    if(v==level) printf("        Bonus point %02d¥n", b*=10*(10+1-c));
                 printf("              Total %02d¥n¥n", h+=(s+b));
    if(v!=level) break;
  }
  print "";
  if(round>last_round) print "Congratulations !!";
  else printf("Game Over  (Result:Round%02d)¥n",round);
}
```

# スクリプトでスクリプトを
# ～ Ruby で PS で迷路 ～

海鳥

## 0. 概要

迷路を描画する PostScript ファイルを作成する Ruby スクリプトを紹介する。

## 1. はじめに

僕の出身の研究室は、数値計算、特にシミュレーションが専門だった。それぞれ研究室には伝統があるものだが、その研究室の伝統は「PostScript が読み書きできるようになる」というものだった。

シミュレーションをしたら、その結果が正しいかどうかをチェックするためにデータの可視化をしなければならない。その際、OpenGL を使うのはなんか大げさで、他の描画ライブラリ(たとえば Windows GDI)なんかでは機種依存があるので面倒だ。そこで、手軽な可視化手段として PostScript が良く用いられた。PostScript なら単にテキストファイルを吐くだけで良いから機種依存性はないし、論文を書くのに使われる LaTeX との相性も良い。

そんなわけで、研究室のメンバーは別に強制されたわけでもないのに、先輩が使っているのを見ているうちに自然と PostScript が読み書きできるようになっていった。研究室に入るまでプログラムをほとんどやったことが無かった「いたいけな」女の子も、数年もすると図を描くのに Cygwin 上の vi で PostScript を生書きするようになっている、そんな研究室だった。

現在、僕はシミュレーションを仕事としているが、まだ PostScript を日常的に良く使っている。PostScript で連番のファイルを吐いておいて、ImageMagik の Convert で Windows Bitmap に一括変換、エンコーダに放り込んで MPEG ムービーを作るというコンボは非常に便利だし、別のソフトが吐いた図をちょっと修正したりするのに PostScript の知識は欠かせない。

そんななか、TSNET スクリプト通信に僕の４コマを掲載させていただくことになった。拝見すると、Python や AWK の話題が載っている。それを読んでみて、プログラマの端くれとしてなんか記事を投稿してみたくなった。僕がメインに使うスクリプト言語は Ruby であるが、いまさら Ruby で何か書いてもあまりインパクトはないだろう。

では、PostScript の記事はどうだろうか。PostScript もスクリプト言語であるわりに、あまり TSNET で触れられることは無いようだ。

そんなわけで、本稿では PostScript を使ったお遊びプログラムを一つ、紹介してみたいと思う。

## 2. プログラム言語としての PostScript

PostScript(以下 PS と略す)を JPEG や PNG といったイメージ形式の一つと思っている人は多い。

しかし、Scriptという名前からわかるようにPSはプログラム言語なのである。プログラム言語だから条件分岐やループ処理なども備え、基本的にはなんでもできる。スタックマシンに基づいたシンプルな文法は個人的に結構好きなのだが、逆ポーランド記法を用いることからコードがちょっと読みづらく、それで敬遠されている(ような気がする)。

PSは、もともとプリンタのために開発されただけあって、ベクタグラフィックを描くための強力な関数群が多数用意されている。また、特に局所座標と実座標を分けることで、ゲームで言うところのスプライトのようなものが実現でき、複雑な図が簡単なコードでかけてしまう。しかし、言語としてみたPSは
・変数が無い(データはすべて共通のスタックを用いる)、
・構造化されていない(がんばればできないことは無いが)、
・名前空間が基本的にグローバルしかない(そもそも名前空間という概念が無い)
・マクロが単純な文字列展開なので型チェックなどが無い(そもそも型が文字列と数値しかない)
といった感じで、本格的なコードを書くにはちょっとしんどいところがある。

そんなわけで、描画部分はPSで行うが、他の処理はRubyに任せることにしよう。全部PSでやったほうが企画としては面白いがあまり他の人の役に立たないし、何より「スクリプト言語で別のスクリプト言語を出力する」というのがなんともメタな感じでよいではないか。

## 3. スクリプトの解説

前置きが長くなったが、スクリプトの内容を簡単に解説しよう。スクリプトの本体はmaze.rbというRubyスクリプトである。Mazeというクラスがあり、コンストラクタでサイズを受け取る。迷路作成には「クラスタリングアルゴリズム」という手法を用いている。詳しくは同じアルゴリズムを使う拙作ソフト「迷次郎」の解説ページを参照されたい。
http://homepage1.nifty.com/kaityo/maze/algorithm.html

Mazeクラスは、コンストラクタで受け取ったサイズを短辺としてA4サイズ一杯になるように長辺のサイズを決めて迷路を作成し、PSとして出力する。デフォルトでの大きさは50としてある。

exportEPSというメソッドがPSファイル、正確にはEPSを出力するためのメソッドである。EPSとはEncapsulated PostScriptの略で、PSファイルにコメントの形でサイズなどの付加情報を埋め込んである。そのうち、特に重要な情報はBoundingBoxである。PSの描画空間は基本的に無限大である。他のソフトに埋め込むには、その描画空間のうちどこからどこまでを切り取って表示するかを指定しなくてはならない。その切り取る長方形の座標がBoundingBoxである。今回は(0,0,600,850)、すなわちA4サイズに固定しておき、
作成する迷路がちょうどA4サイズとなるようにコンストラクタでグリッドサイズ(@GridSize)を決める。

PSは数多くの描画コマンドを持つが、直線を引くだけなら単純である。多くの描画システムと同様に、PSもカレントポイント(現在の筆の位置)を持っている。したがって(x1,y1)から(x2,y2)に線を引きたければ、まずカレントポイントを(x1,y1)に移動し、そこから(x2,y2)に線を引けばよい。PostScriptは逆ポーランド記法なので、

```
x1 y1 moveto
x2 y2 lineto stroke
```

と書く。strokeは、「これまで書いた仮想的なパスを実際に描画せよ」という命令で、これも奥が深いが、とりあえず深入りはしないでおこう。詳しくはPSの赤本、青本を参照して欲しい。

描画は基本的にはこれだけで良いのだが、せっかくなのでもう少しPostScriptのプログラム言語らしさに触れよう。迷路の入り口と出口に矢印を描画することにして、その矢印をPostScriptのマクロにより描画しよう。

PostScriptのマクロ定義は

```
/macroname {中身} /def
```

である。以後、 macroname とあったら、定義された中身で展開される。マクロの中で別のマクロを使うこともできるし、実行時に定義されていれば、定義時には定義されていないマクロを使うことも可能だ。

そこで、矢印のマクロ arrowを以下のように定義する(グリッドサイズが2の場合)。

```
/arrow {gsave translate 0 0 moveto 2 0 lineto stroke
2 0  moveto 1.0 -1.0 lineto 1.0 1.0 lineto closepath fill stroke grestore} def
```

最初と最後のgsaveとgrestoreは、描画の情報(カレントポイントなどのシステム情報)を覚えておいて、最後に復帰するための命令である。

次に、いきなり translateとあるのは、原点の移動である。このマクロが呼ばれる前に二つの数値がスタックに積まれていることを想定している。つまり、

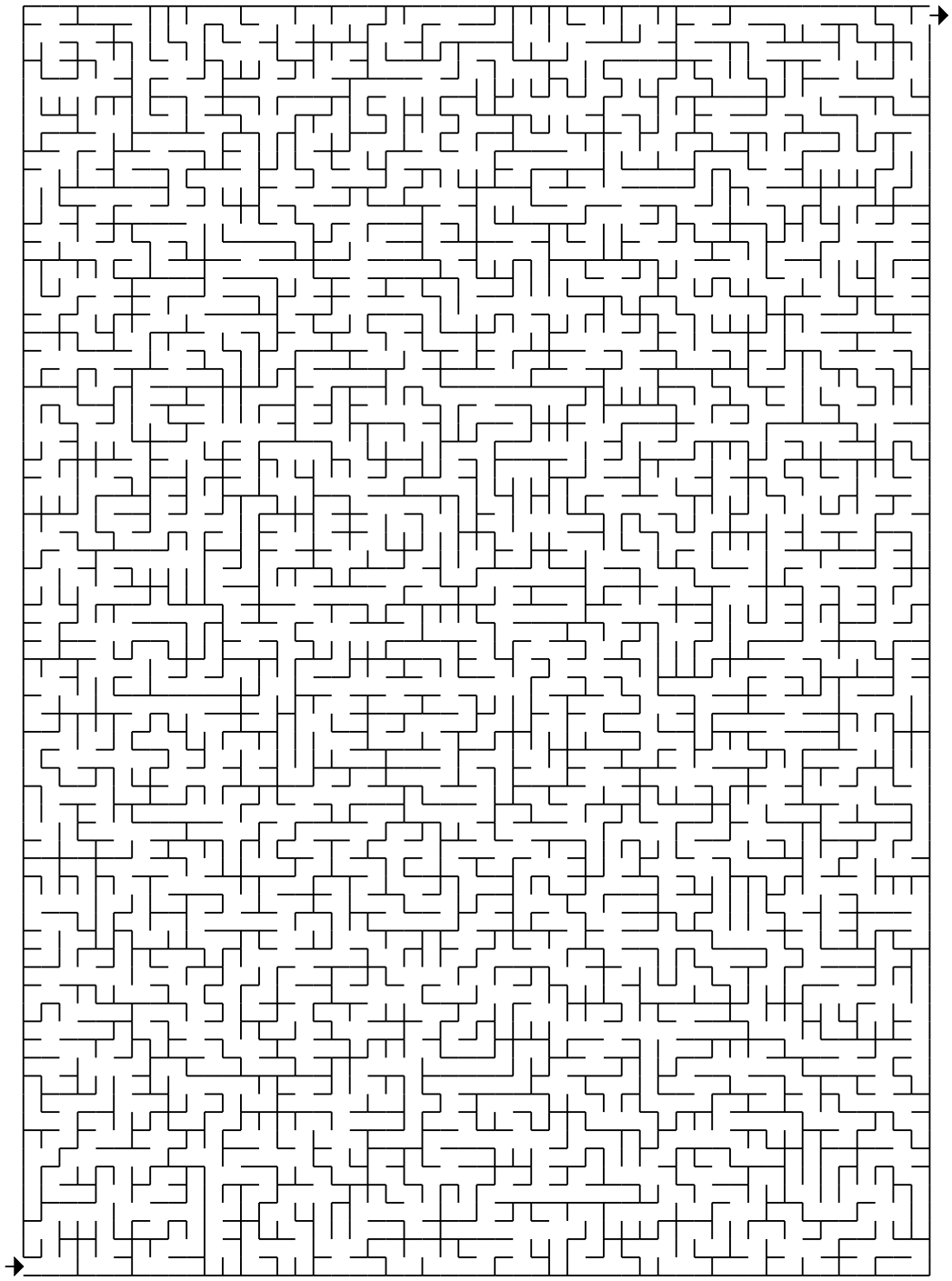
```
x y arrow
```

とすると、座標(x,y)を原点として描画する。この原点移動はgrestoreによって元に戻るため、他の描画には影響を与えない。

あとは迷路の入り口(左下に固定)と出口(右上に固定)の座標を書いて arrowとすれば矢印が書かれる。

## 4. 実行結果

デフォルトの実行結果を次のページに貼っておこう。A4一枚にまるまる書かれた迷路が出てくるのはちょっと愉快である。



## 5. 終わりに

以上、迷路を描画するPostScriptを出力するRubyスクリプトを紹介してみた。スクリプトがスクリプトを吐く、というメタな関係は面白い。最初にこの関係に気が付いたのは、PerlでCGIを組んでいたとき、出力されるHTMLにJavaScriptが含まれていたときだ。PerlがJavaScriptを吐き、そのJavaScriptを使って作られたデータがまたPerlにPostされる。このメタな関係を使って何か面白そうなことができないかと考えているが、まだ何も思いつかない。

メタといえば、makeも結構メタである。手元のプロジェクトはほとんどがトップレベルにGNU makeが君臨している。makeすると、rubyスクリプトが実行され、出力されたコードをg++がコンパイルし・・・みたいなことは日常茶飯事であった。良く考えればスクリプトが担う仕事は、そのほとんどが単独ではなく、他の言語と連携していることが多い。そういう意味ではスクリプト言語はメタそのものかもしれぬ。

今回のスクリプトでは「PostScriptらしさ」をあまり使わなかった。PostScriptはもっと楽しい言語である。興味のある方は、サンプルスクリプトを修正してカラーの迷路を出してみたり、そもそもRubyを使わずフルPSで書いてみたりしてみればきっと楽しいことであろう。

最後に参考文献を挙げておく。

青本 - 「PostScript チュートリアル&クックブック」 AdobeSystems著 ASCII出版局
PSの初心者はまずこれを読むこと。PSを知っておいて損はない。

赤本 - 「PostScript リファレンスマニュアル(第２版)」 AdobeSystems著 ASCII出版局
PSを日常的にいじるようになったら、赤本があったほうが便利。

・・・といいつつ、この通信の読者の棚には高確率で上記の本は入っていることだろうが。

[maze.rb]

```
# maze.rb
# Author: Kaityo
class Maze
  def initialize(s)
    @lx = s
    @ly = @lx*297/210
    @bond_h = Array.new((@lx+1)*@ly) { false}
    @bond_v = Array.new(@lx*(@ly+1)) { false}
    @point = Array.new(@lx*@ly*2) {|i| i}
    @GridSize = (600.0-30)/(@lx+4)
    @LeftMargin = @GridSize*3
    @TopMargin = @GridSize*2
    makeMaze
  end

  def getClasterIndex(x, y)
    index = @lx*y+x
```

```
    while(index != @point[index])
      index = @point[index]
    end
    return index
  end

  def connect(ix1, iy1, ix2, iy2)
    i1 = getClasterIndex(ix1,iy1)
    i2 = getClasterIndex(ix2,iy2)
    if i1<i2
      @point[i2] = i1
    else
      @point[i1] = i2
    end
  end

  def makeMazeSub
    rate = 0.8
    for ix in 0..@lx-2
      for iy in 0..@ly-1
        next if rand <rate
        next if getClasterIndex(ix,iy) == getClasterIndex(ix+1,iy)
        @bond_h[(@lx+1)*iy+ix+1] = true
        connect(ix,iy,ix+1,iy)
      end
    end
    for ix in 0..@lx-1
      for iy in 0..@ly-2
        next if rand <rate
        next if getClasterIndex(ix,iy) == getClasterIndex(ix,iy+1)
        @bond_v[(iy+1)*@lx+ix] = true
        connect(ix,iy,ix,iy+1)
      end
    end
  end

  def makeMazeFinal
    for ix in 1..@lx-2
      for iy in 1..@ly-1
        next if getClasterIndex(ix,iy) == getClasterIndex(ix+1,iy)
        @bond_h[iy*(@lx+1)+ix+1] = true
        connect(ix,iy,ix+1,iy)
      end
    end
  end

  def makeMaze
```

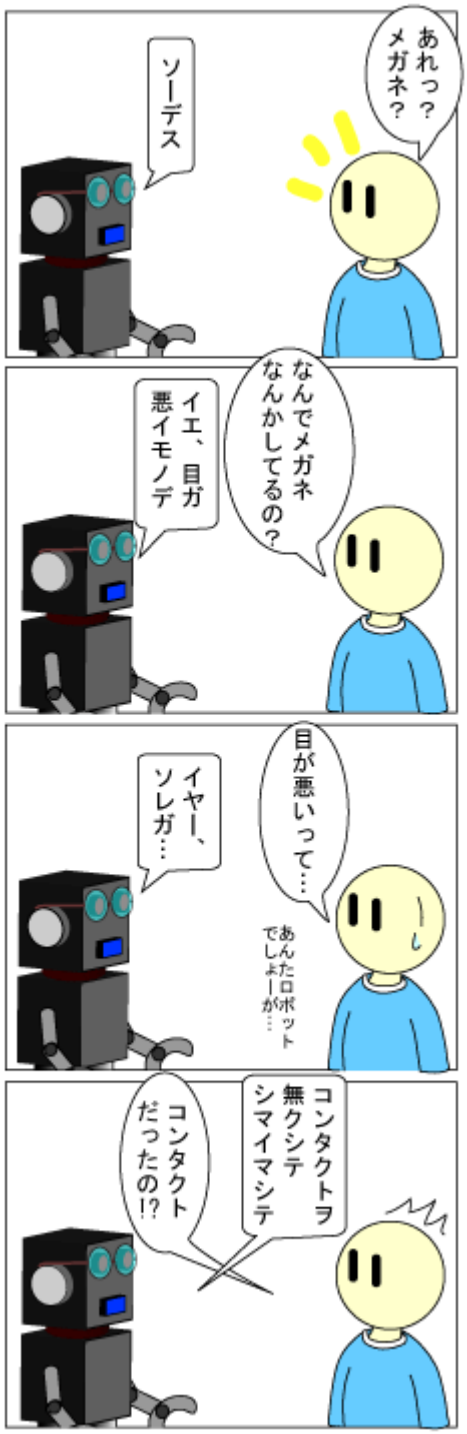
```
    for i in 0..10
      makeMazeSub
    end
    makeMazeFinal
    @bond_h[0] = true
    @bond_h[(@lx+1)*@ly-1] = true
  end

  def exportEPS (filename)
    f = open(filename,"w")
    f.print "%!PS-Adobe-2.0¥n"
    f.print "%%BoundingBox: 0 0 600 850¥n"
    f.print "%%EndComments¥n"
    f.print "/mydict 120 dict def¥n"
    f.print "mydict begin¥n"
    f.print "gsave¥n"

    g = @GridSize
    h = @GridSize*0.5
    f.print "/arrow {gsave translate "
    f.print "0 0 moveto "
    f.print g.to_s + " 0 lineto stroke¥n"
    f.print g.to_s + " 0  moveto "
    f.print h.to_s + " -" + h.to_s + " lineto "
    f.print h.to_s + " " + h.to_s + " lineto "
    f.print "closepath fill stroke grestore} def¥n"

    f.print @LeftMargin.to_s + " " + @TopMargin.to_s + " translate ¥n"
    f.print "-" + g.to_s + " " + (g*0.5).to_s + " arrow ¥n"
    f.print "" + (g*@lx).to_s + " " + (g*(@ly-0.5)).to_s + " arrow ¥n"

    for ix in 0..@lx
      for iy in 0..@ly-1
        x = ix * @GridSize
        y = iy * @GridSize
        next if @bond_h[iy*(@lx+1)+ix]
        f.print x.to_s + " " + y.to_s + " moveto "
        f.print x.to_s + " " + (y+@GridSize).to_s + " lineto stroke¥n"
      end
    end
    for ix in 0..@lx-1
      for iy in 0..@ly
        x = ix * @GridSize
        y = iy * @GridSize
        next if @bond_v[iy*@lx+ix]
        f.print x.to_s + " " + y.to_s + " moveto "
        f.print ""+(x+@GridSize).to_s + " " + y.to_s + " lineto stroke¥n"
```

```
      end
    end
    f.print "grestore¥n"
    f.print "end¥n"

  end
end

s = 50
if(ARGV.size > 0)
  s = ARGV[0].to_i
end

m = Maze.new(s).exportEPS("maze.eps")
```

# よしおさんとロボ太

海鳥作

「メガネロボ」

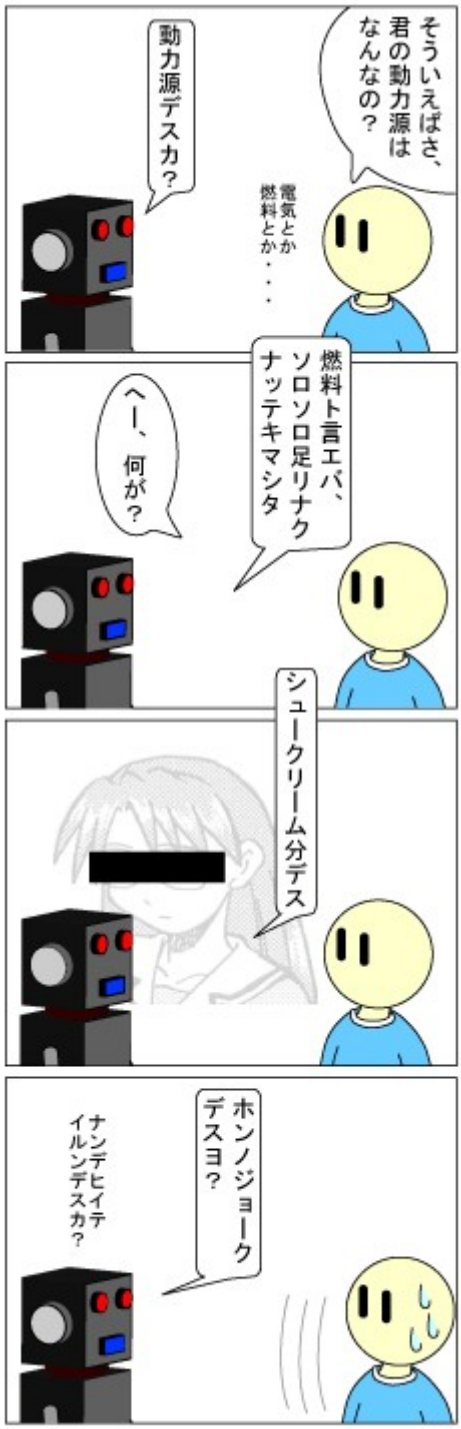



メガネっ子(言い切った)

「続・メガネロボ」


拡散ロボ粒子砲

「ロボ動力」




勉強とかすると減ります。

# PerlMagick & Cooliris入門

jscripter

## I. はじめに

「Windowsユーザーのための ImageMagick超入門」を書いたのは2001年のことだ。その頃はまだデジタル写真は著者の「更新日記」には存在せず、TS Networkのロゴを変形させて遊んだに過ぎなかった。実用的な用途がなかったために、入門がさらに発展することはなかった。その後、デジタル写真はデスクトップのリソースとして、またWebのコンテンツとしても重要な位置を占めはじめたことは間違いない。本稿ではImageMagick/PerlMagickによる画像の回転と縮小およびモンタージュの作成の方法を示し、作成した適当なサイズの画像とサムネイルを使って、ホームページの画像をCoolirisで一覧する方法を紹介する。

Coolirisは、ホームページの画像や動画などをインタラクティヴに一覧・拡大縮小、スライドショー形式で見ることが可能なWebブラウザ・プラグインである。

## II. デジタル写真管理・ウェブパブリッシングシステムの構想

GoogleのPicasaをはじめとして、携帯電話やデジタルカメラに添付されるデジタル写真・画像管理プログラムが機種を変えるたびに多数、デスクトップにインストールされているのが実情だが、いずれも画像の管理の仕方が異なり、実用的にこれならというものがない。著者はPicasaを使用しているが、ホームページで画像を利用する場合のシステム化などは想定されていない。もともとそのような使い方を想定していないアプリケーションと言えばそれまでである。そのような場合には自前のデジタル写真管理・ウェブパブリッシングシステムを構想してみるのも悪くない。いずれはデスクトップCGIとしてまとめるが、本稿では、要素技術としてのPerlMagickとその応用を考えてみよう。

## III. ImageMagick/PerlMagickとCoolirisのインストール

2001年以来、何度かImageMagick/PerlMagickのインストールを試みたはずだが、あまり良い記憶がない。今回は運良く、PerlMagickはActivePerl 5.10.0.1004用のパッケージとなっていたので、安心してインストールできた。したがって、Perlは必然的に「ActiveState Perl v5.10.0 build 1004」を使うことになる。Perl5.8をメインで使っている場合には、別のディレクトリにインストールしておこう。

### ImageMagickとPerlMagickのインストール

まず、

ImageMagick: Install from Binary Distribution
http://www.imagemagick.org/script/binary-releases.php#windows

のWindows用バイナリの配布ページから、

ImageMagick-6.4.8-10-Q16-windows-dll.exe

をダウンロードして実行し、インストールしよう。インストールの過程で「install PerlMagick for ActiveState Perl v5.10.0 build 1004」をチェックすること。インストーラの指示に従って、コマンドプロンプトからインストールできているかどうかを確認しておく。

Cooliris のインストール

Cooliris はFirefox、Safari、Internet Explorer でサポートされている。下記のサイトからプラグインを入手してインストールしよう。

Cooliris | Discover More
http://www.cooliris.com/

## IV. ImageMagick/PerlMagick による画像処理

デスクトップでデジタルカメラ画像を取り扱う場合、横や逆さまになった画像を回転させることは重要だが、数多くの画像を取り扱う場合には大変煩雑な作業となる。自動的に適切に回転してくれないものかと思うことが多い。いずれ、カメラに加速度センサーや地磁気センサーが内蔵され、カメラをどのように回転させ、どの方向に向けて撮影しているかの情報がデジタル写真に取り込まれるようになれば、自動的な画像変換が行われるようになるかもしれない。それまでは、Gimp やPhotoshop などで回転加工する必要があるだろう。

本稿では、Explorer でサムネイルを見ながら、ファイル名に回転させたい角度の情報を付加して、PerlMagick を使用して一括して回転加工する。さらに適度なサイズに縮小し、サムネイル用の画像を生成する。さらに縮小して(必要ならばでよいのだが)、縮小画像を並べたモンタージュを生成する。そのような画像処理を特定のディレクトリに置いた複数の画像に適用するスクリプトを書いてみよう。スクリプトとしては大変初歩的なものである。便宜上、画像ファイルは JPEG 形式のものであること、スクリプトを実行するのは画像が置かれているディレクトリであることを想定している。

画像を回転させる

画像ファイル名に、右に 90° 回転させる場合には、「_r1」を付加しておく。180° 、270° ならば、それぞれ、「_r2」、「_r3」を最後に(.拡張子の前に)付加する。ファイル名を読んで、回転させるメソッドを適用するスクリプトである。回転させた画像のファイル名には「o」を付加して出力する。「_r」のあとにある数字が 1,2,3 以外の数字の場合は何もしない。

[image2rot.pl]

```
#!/Perl5.10/bin/perl
use Image::Magick;
#
# Get image file names.
#
opendir(DIR,".");
```

```
@imagefiles = grep(/^¥w+(_r¥d)*¥.jpg$/i, readdir(DIR));
close(DIR);
#
# Create thumbnails.
#
foreach $imagefile (sort @imagefiles){
	($rotfile = $imagefile) =~ s/¥.jpg/o¥.jpg/i;
	$image=Image::Magick->new();
	$x=$image->ReadImage("$imagefile");
	warn "$x" if "$x";
	$image->Set(background=>'white');
	$copy=$image->Clone();
	$copy->Set(page=>'0x0+0+0');
	if($imagefile =~ /_r(¥d)¥.jpg/i){
		if($1 > 0 && $1 < 4){
			$copy->Rotate(90*$1);
		}else{
			next;
		}
	}
	$copy->Write($rotfile);
}
```

---

このスクリプトで得られる画像ファイルをCoolirisで表示する標準画像として用いる。

画像を1/4(25%)に縮小する

ファイル名が/^¥w+_r¥d¥.jpg$/iにマッチしない、すなわち回転指定がされている元ファイルを除いたファイルを1/4に縮小し、ファイル名に「_qs」を付加して出力する。回転指定しているファイルを縮小することは不要だからである。回転させた画像は縮小する。この画像をCoolirisで表示するサムネイルとして使用する。

[image2qs.pl]

---

```
#!/Perl5.10/bin/perl
use Image::Magick;
#
# Get image file names.
#
opendir(DIR,".");
@imagefiles = grep(!/^¥w+_r¥d¥.jpg$/i && -f, readdir(DIR));
close(DIR);
#
# Create thumbnails.
#
```

```
foreach $imagefile (sort @imagefiles){
       ($qsfile = $imagefile) =~ s/¥.jpg/_qs¥.jpg/i;
       $image=Image::Magick->new();
       $x=$image->ReadImage("$imagefile");
       warn "$x" if "$x";
       $image->Set(background=>'white');
       $copy=$image->Clone();
       $copy->Set(page=>'0x0+0+0');
       $copy->AdaptiveResize('25%');
       $copy->Write($qsfile);
}
```

---

縮小画像をさらに1/4に縮小して並べてモンタージュする

画像ファイル名が/_qs¥.jpg$/iにマッチする、すなわち1/4に縮小したファイルのみを処理する。

[image2montage.pl]

---

```
#!/Perl5.10/bin/perl
use Image::Magick;
#
# Get image file names.
#
opendir(DIR, ".");
@imagefiles = grep(/_qs¥.jpg$/i,readdir(DIR));
closedir(DIR);
#
# Read and reduce images.
#
$images=Image::Magick->new();
foreach $imagefile (sort @imagefiles){
       print $imagefile,"¥n";
       $image=Image::Magick->new();
       $x=$image->ReadImage($imagefile);
       warn "$x" if "$x";
       $copy=$image->Clone();
       $copy->Set(page=>'0x0+0+0');
       $copy->AdaptiveResize('25%');
       push(@$images,$copy);
}
#
# Create image montage.
#
```

```
print "Montage...¥n";
$montage=$images->Montage(geometry=>'80x80+10+5>',gravity=>'Center',
  tile=>'5x',compose=>'over',background=>'#ffffff',
  font=>'Generic.ttf',pointsize=>18,fill=>'#600',stroke=>'none',
  shadow=>'true');
print "Write...¥n";
$montage->Set(matte=>'false');
$montage->Write('montage.jpg');
print "Display...¥n";
$montage->Write('win:');
```

以上のスクリプトでおもしろかったのは、$images というオブジェクト(リファレンス)を @$images として配列にデリファレンスして、$copy というオブジェクトを次々格納していくところ。なるほど、オブジェクト指向はこんな具合に使うものなのかと思った。しかし、オブジェクトの構造を知っていなければ使うことは難しい。オブジェクトは情報を隠蔽しすぎると思う。この部分は PerlMagick で配布されている demo の demo.pl にある。詳細は下記ディレクトリの demo.pl を参照すること。

C:¥Program Files¥ImageMagick-6.4.8-Q16¥PerlMagick¥demo

demo.pl を実行して参考にすれば、PerlMagick/ImageMagick で何ができるか、どのように書けばよいか大体わかるようになっている。モンタージュの実行結果の一例を載せておこう。

図 1 PerlMagick によるモンタージュ写真例

## V. Cooliris と MediaRSS

Cooliris は「クールな虹彩」の意味を持つクールなアプリケーションである。ページのヘッダに link タグで MediaRSS のフォーマットで記載した XML ファイルを指定しておけば、ホームページにある画像ファイルをマウスでインタラクティヴに操作して左右に動かしたり、縮小・拡大して

鑑賞することができる。また、スライドショー形式で観ることが可能になる。

ページ・ヘッダの link タグの記述は例えば、次のようになる。

---

```
<link rel="alternate" href="http://homepage1.nifty.com/kazuf/renewal_media_2009_01.xml"
type="application/rss+xml" title="" id="gallery" />
```

---

MediaRSS に準拠した XML ファイルは、例えば次のようになる。item は一部省略している。

---

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss">
<channel>
	<title></title>
	<link>http://homepage1.nifty.com/kazuf/renewal.html</link>
	<description>マイ・ホームページ</description>
<item>
	<title>SA360080_r1o.jpg</title>
	<description>Walking out to my town at night on January.</description>
	<link>http://homepage1.nifty.com/kazuf/images/2009/01/SA360080_r1o.jpg</link>
	<guid>http://homepage1.nifty.com/kazuf/images/2009/01/SA360080_r1o.jpg</guid>
	<media:description>Going into town at night in January.</media:description>
	<media:thumbnail url="images/2009/01/SA360080_r1o_qs.jpg" />
	<media:content url="images/2009/01/SA360080_r1o.jpg" type="image/jpeg" />
</item>
<item>
	<title>SA360081_r1o.jpg</title>
	<link>http://homepage1.nifty.com/kazuf/images/2009/01/SA360081_r1o.jpg</link>
	<guid>http://homepage1.nifty.com/kazuf/images/2009/01/SA360081_r1o.jpg</guid>
	<media:thumbnail url="images/2009/01/SA360081_r1o_qs.jpg" />
	<media:content url="images/2009/01/SA360081_r1o.jpg" type="image/jpeg" />
</item>
.....
</channel>
</rss>
```

---

MediaRSS は、RSS 2.0 を拡張した Yahoo! search の仕様である。次の URL を参照しよう。

http://search.yahoo.com/mrss

画像を表示するだけなら、media:thumbnail と media:content の url 属性に、それぞれ、サムネイルと標準画像のパスを記載すればよい。

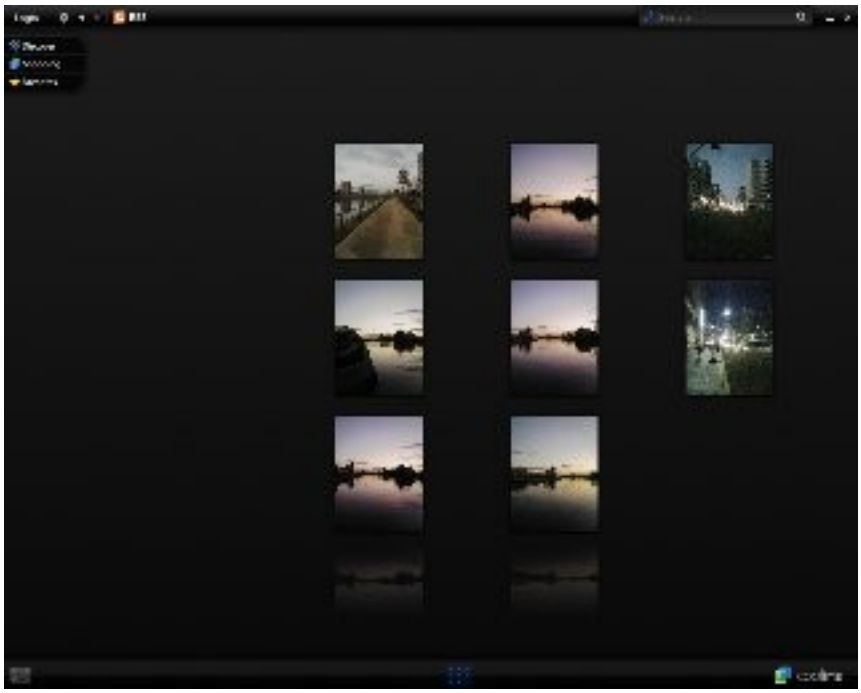


図 2. Cooliris でホームページのサムネイル画像を表示する

media:description は今のところ日本語表示はサポートされていないが、記載しておけば、画像の説明が拡大画像のタイトルの後に：を介して表示される。サポートに日本語表示の希望を伝えると日本語表示への対応も今後の予定に入っているそうだ。期待して待とう。

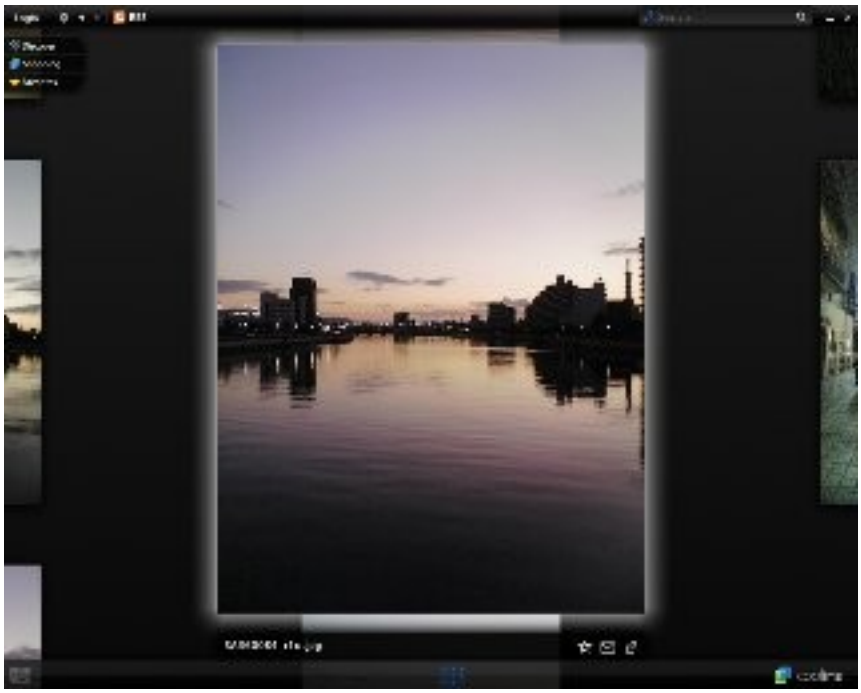


図 3. Cooliris で表示したサムネイル画像を拡大表示する

Media RSSを自動生成するスクリプトを載せておこう。まだ、media:descriptionの記載は考慮していないが、CGI化する際にはそのような機能も付け加えるべきだろう。itemのtitleにも画像ファイル名を取り込んでいるが、別途適切なタイトルを設定できるようにすべきだろう。

[image2mediarss.pl]

```
#!/Perl5.10/bin/perl
#
# Get thumbnail file names.
#
opendir(DIR,".");
@thumbnails = grep(/_qs¥.jpg$/i, readdir(DIR));
close(DIR);

#
# Set your homepage.
#
$baseurl = "http://homepage1.nifty.com/kazuf/";
$mainpage = "renewal.html";
$imagedir = "images/2009/01/";
$description = "マイ・ホームページ";

#
# Media RSS output
#
print <<"END_OF_PRINT";
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rss version="2.0" xmlns:media="http://search.yahoo.com/mrss">
<channel>
	<title>$title</title>
	<link>$baseurl$mainpage</link>
	<description>$description</description>
END_OF_PRINT

foreach $thumbnail (sort @thumbnails){
	($imagefile = $thumbnail) =~ s/_qs¥.jpg/¥.jpg/i;
	print "<item>¥n";
	print "¥t<title>$imagefile</title>¥n";
	print "¥t<link>$baseurl$imagedir$imagefile</link>¥n";
	print "¥t<guid>$baseurl$imagedir$imagefile</guid>¥n";
	print "¥t<media:thumbnail url=¥"$imagedir$thumbnail¥" />¥n";
	print "¥t<media:content url=¥"$imagedir$imagefile¥" type=¥"image/jpeg¥" />¥n";
	print "</item>¥n";
}
print "</channel>¥n";
print "</rss>¥n";
```

## VI. 今後の予定 - 画像処理システムのデスクトップCGI実装

デスクトップCGIフレームワーク開発の一環として、デスクトップに格納されている画像をWebパブリッシングすることも想定する画像加工・管理システムを考えている。画像ディレクトリをスクリプトで読んで、デスクトップ(ローカル)httpサーバーにMediaRSSを生成すれば、Coolirisを画像閲覧システムとして使うことも可能なのである。

既にデスクトップCGIフレームワークでは、RSS/Atomを日記コンテンツの格納に利用し、MySQLデータベースと連携させるために使っているが、画像のメタデータをMediaRSSとして格納して、表示用データとして使えるのは大変便利である。Semantic Webの具体的な成果がパーソナルなレベルでも得られつつあるということができるだろう。現実世界は未曾有の危機に晒されているが、仮想世界には素晴らしい時代が訪れた。

(2009年2月26日)

# 編集後記

jscripter

世の中は快調というわけにはいかないようだが、コンピューティングの世界は絶好調なのかもしれない。なぜそんなことになっているのかには多少残念な気持ちもある。その原因はコンピュータにあったりする可能性もあるからだ。頼りになるのは運だけ、アルゴリズムを頭から信用してはいけない^^;) 景気回復を祈って、まあ今日のところはよしとしよう。Enjoy!

第5号は第2巻第1号、2.1.00x号となり、五月刊行を予定している。

# ＴＳＮＥＴスクリプト通信

2009年3月 2日　　1.4.002改訂版発行
2009年2月28日　　1.4.001版発行

---

## 投稿規程

TSNETWiki ： 「投稿規程」のページを参照のこと

## 編集委員会(投稿順)

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

## 著作権

1. 各記事については、著作者が著作権を保持します。
2. 「TSNETスクリプト通信」の二次著作権は各記事の著作者より構成される編集委員会が保持します。

## 使用許諾・配布条件

1. 編集委員会は「TSNETスクリプト通信 1.4.xxx版」を、ファイル名が「tsc_1.4.xxx.pdf」のPDFファイルとして無償で配布します。また、ファイル名、ファイル内容を一切改変しない状態での電子的再配布および印刷による再配布を無償で許諾します。
2. 関連するスクリプトファイルについては、使用および再配布を無償で許諾しますが、改変後の再配布についてはオリジナルの著作権を併記することを条件に無償で許諾します。
3. 記事およびスクリプトファイル等に著作者の使用許諾・配布条件の記載がある場合は、著作権の項および上記2項に優先するものとします。

## 免責事項

「TSNETスクリプト通信」の内容および同時に配布されるスクリプトなどの使用は、すべて使用者の自己責任によるものとし、使用によって生ずる一切の結果等について、編集委員会および著作者は責任を負いません。

## 編集ソフトウェア

OpenOffice.org 3.0 Writer

## 発行所(一次配布所)

TSNETWiki ： 「TSNETスクリプト通信」のページを参照のこと

---