TSNET スクリプト通信

TSC 編集委員会

目次

巻頭言

jscripter

世は政権選択選挙の真っ最中、TSNET スクリプト通信第 6 号、夏バージョン 2.2 を無事送り出す。

機械さんの「やｎでレ Python ～λ の使徒編～」第 4 回、第 5 回と「お呼びとあらば即参上」の前後編を合わせて全 39 ページ掲載。たっぷりとお楽しみください。

Y ささんは、書き下ろし新作 AWK スクリプト、古典的脳トレ「神経衰弱」です。ゲームの方法は直感的でわかりやすいのですが、記憶力に優れたコンピュータは強敵かも。

海鳥さんは、「よしおさんとロボ太」夏休み短期集中連載の新作 5 編を一挙転載させていただくことにしました。是非話の続きを読みたいと思うのですが^^;)次はどうなるんだろう。

私は「Zed 入門 II - Twitter on Zed」を掲載しました。第 3 号に書いた「Zed 入門」の続編として、Zed のその後と、最近オバマ大統領他の有名人が使ったり、フィジカル・コンピューティングへの応用が話題になっている Twitter を Zed から使ってみたレポートです。例によって、Perl を使ったスクリプトを Zed で動かしています。Perl スクリプティングの観点からは、Net::Twitter を始め、様々なモジュールを紹介、さらに正規表現のオプション sx の使い方について説明しています。

表紙の写真は、昨年と同様、広島宇品の花火大会の写真になってしまいました。ちょっと変った写真を選んでみました。表紙写真投稿歓迎です。

次号、第 7 号は 11 月を予定しています。お楽しみに。

（投稿受付：2009年8月22日）

やｎでレPython
～λの使徒編～

Voice Over
This man is Ernest Scribbler... writer of jokes. In a few moments, he will have written the funniest joke in the world... and, as a consequence, he will die ... laughing.

声：この男はアーネスト・スクリブラー、ジョークライターだ。数分後、彼は世界最高のジョークを書き上げ、そのために死んでしまう。笑い死ぬのだ。

『Monty Python's Flying Circus "The funniest joke in the world"』

■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

第四回「お呼びとあらば即参上　＜前編＞」

　マナブの元に、奇妙なメールが届いたのは、イチコとの甘く危険な(←生命が)プログラミング個人授業の四回目の直前の日のことだった。
　差出人は『λの使徒』。
「入(にゅう)の使徒？」
　意味不明な不気味な名称に、マナブはまず何より気持ちの悪さを覚えた。
（『入』ってなんだろ。入信でもしろってコトかな。でも、何に？）
　スパムメールかな、と思い、即刻削除しようと思ったのだが、タイトルがどうも気にかかり、結局開いて確かめることにした。
　なぜならタイトルは……

『基本三構造の学習を終えたあなたに』

とあったからだ。
　偶然にしてはできすぎている。
　マナブがプログラミングの学習をしていることは、マナブとイチコ以外にも薄々知られている。
　学校で友人に、イチコが打ち出した手作りの教科書を見ているところを聞かれて、そのようなコトを話したことがある。
　また、マナブとイチコの関係についても、どういうつきあいなのかは不明だが、とりあえず友人だ、というところも周知の事実となている。
　しかし、勉強の具体的な進捗状況など、マナブとイチコ以外、誰も知らない筈だった。
　マナブがこのごろよく相談する父にも、詳細を話したことはない。
　となると、イチコのイタズラ、という線が一番現実的なのだが……
（イチコさんがこんなこと、するかなぁ？）
というのが、マナブの率直な感想だった。
　とりあえず、内容を見てみる。

『拝啓
　はじめまして。我々は関数プログラミングを至上のものとし、伝導する、λの使徒です。あなたは現在、手続き型プログラミングの三構造の学習を終え、これから崇高なる関数の世界に入ろうとしています。

（確かにイチコさんは「次は関数だよ」とは言ってたけど……）

『ここで我々はあなたに、一つ御注進差し上げたいと存じます。それは「偽物の関数にご注意を」ということです。

（に、偽物って一体）

『関数とは何か、それは「対応」です。このことは多分あなたも、中学校の数学で習った筈ですが、もしかしたらもう忘れられてしまっているかもしれませんので、蛇足ながら確認させていただきました。

　事実、マナブは忘れていた。

『例えば、f(x)はxの関数ですから、xが決まれば、f(x)は一つの値に決まらなければ対応とは言いません。勿論、f(x)の同じ値が別のxに対応していることについては一向にかまいませんが、同じxで2個以上のf(x)の値が決まるようなら、それは関数ではありません。

（わ、わかんないよ……）

『つまりこうです。x→f(x)で、5→10と7→10は同時に成立できますが、5→10と5→20のように、同じ5に対して違うf(x)が成立したらダメ、ということです。

（あ、そーいうこと）

『全てをシンプルで美しい関数で記述することこそ、関数プログラミングの神髄です。そのためには、まず真の関数を見分ける方法が必要なのです。

（でも、なんで関数なの？）

『あなたは今「なぜそのように関数にこだわるのか？」と疑問を抱いたことでしょう。

（……）

『関数プログラミングは、見やすく理解しやすく、そしてそのため間違いを犯しにくいプログラムを書くための、最上の方法なのです。詳細については、また追って詳しくお伝えいたします。飛ばしすぎ、詰め込みすぎは良くない、ですよね。それでは

敬具』

（……僕らの会話、盗聴されてるのかな？）
　マナブは薄ら寒いものを感じた。

「**にゅうのしと？**」
イチコは怪訝な顔でマナブをみた。
「ま、まさか、マナブくんって**きょにゅう崇拝者**だったの？」
「きょにゅう？　なにそれ……って、ちょっと、いきなり**呪殺モード**になるのやめてよっ！」
誤解で殺されてはたまらない、と、いきなり満身の殺意を漲らせたイチコをなだめにかかる。
しかし、イチコは止まらない。
「マナブくん、胸が大きい子のほうがいいんだ」
「あっ！」
ようやく、意味を飲み込む。
（な、なんて間違いをっ！）
小柄なイチコは、体型もどちらかというと**スリム**だった。
ウェイトが気になる女の子からは、寧ろ羨ましがられているのだが、そこはそこ、人間はどうも隣の庭の芝が青く見えるモノらしい。
それに気づき、あわててマナブは訂正する。
「違う違うっ！　牛乳の『にゅう』じゃなくって、代入の『にゅう』！」
「だいにゅう……おおきいおっ……」
「**あさいんめんとっ！**」
思わず大声を出してしまった後、あわてて周囲を見る。
今は始業前。
担任はまだ来ていないが、遅刻以外のクラスの生徒は、ほぼ来ている。
その全員の視線が、一斉にこちらに集まっている。
実質時間は大して長くなかったが、マナブにとっては無限に続く苦行のように感じられた。
「お前ら、痴話喧嘩を学校に持ってくるなよな」
マナブの前の席のトオル(石川徹)が、呆れ顔で言う。
席の位置の関係から、マナブとも時々話す友人だが、同時にそれは、イチコとの会話を間近で見せつけられ続けているということに等しい。
マナブにとっては、実は文字通り**死活問題**なのだが、なにかそれを言うと一層誤解されそうな気がして、口を閉じて曖昧に笑った。
……というか、痴話喧嘩？
「えっと……痴話喧嘩って、確か、恋人同士とかでやるもんだよね？」
「そーだよ。お前らつき合ってるんだろ？」
当然のようなロぶりに、マナブとイチコの顔色が、同時に真っ赤になる。
当人たちの認識を余所に、周囲ではコレが周知の事実となっていたのだ。
「えっと、その……あの……」
真っ赤になって俯いているイチコはさておき、一度赤くなった顔を青くして冷や汗をかいて口ごもっているマナブに、トオルは首を傾げる。
「はて、定番だとここで『つき合ってなんかないっ！』とか口走って、修羅場になるんじゃね？」
確信犯であった。
「お、お前、わかってて……」
「他人の不幸は蜜の味ってね。毎日毎日朝っぱらから仲いいとこ見せつけられてるんだ。これくらいの茶目っ気は罰は当たらんだろう？」
「……」
理不尽だと思いつつも、言い返す言葉が無い。
「しかしまぁ、反射的に否定しない、ってのはある意味感心だな。少し感動したぞ、うんうん」
（選択肢間違えたら、**デッドエンド**なんだよ）
思っても言えない、マナブだった。

「マナブくん、コレ『にゅう』じゃなくて**ラムダ**だよ」
「え？」
　業後、例によってマナブの家に来たイチコは、例のメールを見て即座に言った。
「ラムダ？」
「そう、ギリシャアルファベットのラムダの小文字。たしかにちょっと見は漢字の『入』に見えるけど、別物だよ」
「知らないよ、そんなの」
　大文字の『Λ』なら、マナブでも知っていたが、小文字を見るのは初めてだった。
「うん。そうだね。でもそれで今日は恥かいちゃったよ」
「……それはゴメン」
「……ところでマナブくん、ホントは**胸の大きい子のほうが好き？**」
　蒸し返すなよ、と内心泣きたいマナブだったが、そこはそこ、既にいつかその話題に触れられる可能性を考えて、答えは用意してあった。
「あのね、イチコさん。体の一部分に固執するヤツはフェチって言うんだよ」
「でも、男の子は大抵……」
「今は色々な趣味があるから、一概には言えないよ。僕は、って言うなら、どっちでもない。す、好きになった女の子の全部がい、一番、だよ。たぶん」
　やはり恥ずかしさに、少しだけ噛んでしまった。
　イチコは、と、見ると、やっぱり赤くなって俯いている。
(だから、こんな話題、振らなきゃいいのに)
「そ、それより、ラムダって何？　数学でΣ(シグマ)とかΔ(デルタ)とか使うのは知ってるけど、ラムダってなんだっけ」
　イチコも慌てて応じる。
「あ、うん。ラムダっていうのは、**ラムダ計算**のラムダだよ」
「ラムダ計算？」
「うん。ちょっと難しい話なんだけど、関数で使うf(x)は、値を表してるか式を表してるか、微妙だよね」
「え？　えっと……」
「y=f(x)って書くとき、yはf(x)、つまり、f(x)で表される式にxを代入した値がyになる、っていう意味だよね」
「う、うん」
「だから、例えばf(x)=x+1と書かれていても、それはf(x)がx+1に等しいわけじゃなくって、x+1のxに何かを代入した結果が、f(x)になるって意味になっちゃうよね」
「う、ううう……う、うん」
　マナブはぐるぐると混乱した末に、ようやく返事をする。
「やっぱり難しかった？」
「えっと、難しいというか、考えたことなかったことだから。でも言われてみれば、確かにそうだよね」
「うん。ところで、式の代入って覚えてる？」
「えっと……」
　数学は苦手なんだよ、と思いながらも必死で記憶を手繰る。
「確か、A=x+y,B=x-yで、AB=(x+y)(x-y)とかいう、アレだよね」
「うんうん、そうそう。その時のAとBって、正しく**『式そのもの』**だったよね」
「言われてみれば」
「そういう『値』じゃなくって**『式そのもの』を表す書き方が、λ(ラムダ)式**なんだよ。今の例で言えば、

A ⇒ λxy.x+y
B ⇒ λxy.x-y

と書けるんだよ」
「……えっとえっと……もしかして、x+1の式自身はλx.x+1でいいの？」
「うん。学校ではf(x)みたいな１変数関数しか習わないけど、プログラミングではf(x,y)やf(x,y,z)みたいな２変数の関数も普通に出てくるんだよ」
「うぁ。なんか難しそう」
「そんなことないよ。例えばf=λxy.x+yとして、(x, y)=(1, 2)のとき、f(x, y)っていくつだと思う？」
　マナブは少し考えて答える。
「もしかして……3？」
「うん。難しい？」
「全然」
　考えたのは主に、簡単すぎてかえって穿ったからだ。
「っていうか、そんなんでいいの？　グラフとか何とか……」
「学校では関数といえば『グラフを書く』がセットになってくるから難しいんだよ。そもそもグラフなんて書かなくてもいいなら関数自体はただの**対応**なんだから、難しいわけないよ」
「対応って……そういえば、にゅ……じゃなかった『λの使徒』ってヒトも、『関数は対応だ』って言ってたなぁ。なんとなくわからないでもないけど、結局どういうこと？」
「つまり、『xの関数』は『xの値が決まれば決まる値』っていう意味だよ。簡単に説明するとね、たとえばマナブくんがおみくじで四十九番を引いたとするよね」
「うん」
「その結果が大吉であれば、49→大吉っていう関数になるんだよ。関数って本来、それだけのモノなんだよ。ある値に対して必ず特定の値が対応する。それが関数」
「そうなんだ」
「ただ普通は、特定の数に対して特定の数が対応する場合を関数として、数以外の一般的なものを対応させるときは**写像**って言うよ。だから関数は、写像の一種、かな？」
「しゃ、しゃぞう？」
「うん。要するに『対応』を数学っぽく言っただけのものだよ」
「そうなんだ」
「ただ、プログラミング言語では、写像と関数は別物を指す場合が多いから注意が必要だよ。もちろんPythonでも、別。というか、プログラミング言語の関数って、名ばかりで全然関数じゃないモノもあるんだよ」
　マナブの脳裏に、メールの内容が甦る。
「あ、『λの使徒』のヒトの言ってた」
「うん。あらゆる意味で関数っぽく無い関数と言えば、今まで出てきた中ではinput関数が最高だよ」
「どういうこと？」
「本当の関数なら、**引数(ひきすう)に取ったものと対応してることが条件**なんだよ」
「ひきすう？」
「関数の後のカッコの中に入れる内容だよ」
「あれ？　でも確か、書かれた内容が表示されるんじゃなかったっけ？」
「表示される内容じゃなくって、戻り値と対応しなきゃならないんだよ」
「も、戻り値？」
「少し説明が必要だね。そういえば、『関数らしい関数』がどんなものか、説明してなかったね」
「うん」
「えっと、今までででてきた関数の中で『関数らしい関数』って、実はlen関数くらいしかやってないんだよ」
「えっと、文字数を数える関数だっけ？」
「本当は文字数だけじゃないんだけど、今は話がややこしくなるからそうしておくね。len関数の使い方って覚えてる？」

「えっと確か……

```
>>> s = 'aiueo'
>>> len(s)
5
>>>
```

こんな感じだっけ？」
「うん。ここで言えば、引数は s、戻り値は 5 だよ。でも、これだと『表示されたもの』と『戻り値』が紛らわしいから、代入してみるね」

```
>>> n = len('aiueo')
>>> n
5
>>>
```

「えっと……」
「前回、『式の評価』の話をしたよね。戻り値っていうのは『評価した結果』なんだよ。つまり、2+3 は評価されて 5 になるから、n=2+3 で n の中に 5 が入るんだよね」
「うん」
「同じように len('aiueo')も評価されて 5 になるんだよ」
「あ、そういうことなんだ」
「図に書くとこんな形かな」

関数(引数) ⇒ 戻り値

「じゃぁ、print 関数とか、そうなんだ」
「違うよ」
「え？」
「うん、実験してみればわかるよ。print 関数に引数として数字の 5 を渡してみて」
「わ、渡す？」
「うん、引数として書くっていう意味。そしてその戻り値を、さっきみたいに『n』に入れてみて」
「うん」

```
>>> n = print(5)
5
>>>
```

「n の中身を見てみて」

```
>>> n
>>>
```

「あれ？　出ないよ？」
「n を print 関数に渡してみて」
「うん」

```
>>> print(n)
None
>>>
```

「のん？」
「英語読みなら、どっちかというと『ナン』かな？」
「何なの、これ？」
「None っていう特殊オブジェクトだよ」
「特殊オブジェクト？」
「うん。『なにでもないもの』っていう意味。要するに『空っぽ』」
「何でそんなヘンなもの……」
「理由は多分、もう少しプログラミングを知ってからじゃないと、理解できないと思う。今必要なの

は、表示される数値が、必ずしも戻り値とは限らない、ということ」
「えっと、最初の話は……そういえばinputだったっけ？　あ、そっか」

```
>>> n = input('IN:')
IN:100
>>> n
'100'
>>>
```

「そう、戻り値は入力によって決まるから。引数⇒戻り値の対応が一切無い、つまり関数っぽくないということだよ。了解？」
「了解」
「じゃ、関数の定義の仕方をやろっか。普通は定義文から入るのが定石なんだけど、せっかくλ式の話をしたんだから、λで書いてみようか」
「お任せするよ」
「ん。それではまず、引数を取って、それに１を加えて返す関数からね」

```
>>> f = lambda x: x + 1
>>> f(10)
11
>>>
```

「らむぶだ？」
「これでラムダって読むんだよ。'b'は無声音。'comb'とか'bomb'とかの'b'と同じだよ」
「要するに、ラムダの後に変数書いて、コロンの後にその式を書けばいいんだね」
「うん。２変数以上の変数はこう書くよ」

```
>>> f2 = lambda x, y: x + y
>>> f2(5, 10)
15
>>>
```

「簡単だね」
「そうだね。計算式を省略するだけなら、これでも十分だよ」
「あ、関数って、変数を代入して評価したら、その数値と入れ替わるんだよね？」
「うん」
「なら、こんなこと、できるかな？」

```
>>> f(5) + f2(4,10)
20
>>>
```

「さっきの『式の計算』から思いついたんだけど、うまくいったね」
「……」
「どうしたのイチコさん？」
「マナブくん、これ、誰かから、やり方聞いた？」
「ううん？　いまさっきイチコさんに聞いたばっかりだけど、なぜ？」
「マナブくん、数学苦手って言ってなかった？」
「うん。苦手だよ」
　イチコは暫く考えた後、
「マナブくん。文系選択志望？」
「えっと……」
　実は、イチコとこうなる前は、数学が嫌いなので理系など論外、と思っていたのだが……
「い、イチコさん、理系だよね」
「うん？」
「だ、だから僕も、で、できれば理系がいいかな、って思ってたんだけど……はは、やっぱ僕の頭じゃ無理だよね」

イチコはマナブの答えに、一瞬、呆然としたが、すぐに今までで最高に柔らかい笑顔で……
マナブを抱き締めていた。

「え？」
女の子の胸にいきなり抱かれて、マナブは頭が沸騰した。
「行こう、一緒に」
「……」
「マナブくん、理系の才能あるよ。ただ、今までの学校の勉強の中では、わからなかっただけだよ」
「そ、そうなの？」
「うん。勿論一緒、って行ってくれたのは嬉しいけど、ホントにマナブくんに才能ないなら、そんな無理は言わないよ。でも、今みたいに、柔軟な頭の使い方ができるのなら、大丈夫だよ」
そして、名残惜しそうに身を引く。
密着していてわからなかったが、イチコの顔もマナブに負けず劣らず真っ赤だった。
しばらく、気恥ずかしいが決して心地の悪くない沈黙がつづいた。
「えっと、あ、あ、あのっ、きゅ、休憩とろっか？」
「……うん」
とはいえ、気恥ずかしさの臨界点を突破したマナブは、逃げるように部屋から出て行った。
毎度おなじみティータイム。
「それにしても、『λの使徒』って一体誰だろう」
そもそもの疑問を口にするマナブに、イチコは少し言いづらそうに切り出す。
「えっと、ちょっと心当たりが……」
「あるの？」
「うん。多分、**あたしのお姉ちゃん**……」
「へ？」
意外な言葉に、一瞬マナブの思考が止まる。
「イチコさんって、姉妹(きょうだい)いたの？」
「うん。お姉ちゃんと妹がいるよ。そういえば話して無かったよね」
「僕は兄弟がいないから、そういう発想無かったなぁ……」
成る程、イチコの姉ならば、詳しい経緯を知ってても不思議はない。
「それにしても、お姉さんに僕のメールアドレス、教えたの？」
珍しくちょっと不満そうに言うと、イチコは青ざめて首を振った。
「言ってないよっ！　で、でも、一度メールを打ってるとこ見られてるから、そのとき覚えたんだと思う」
マナブのメールアドレスは比較的短いので、不可能ではないだろう。
「……ごめん、疑ったりして」
「い、いいよ。あたしが迂闊だったんだから。それにあたしにプログラム教えてくれたの、お姉ちゃんだから、マナブくんに話す内容も、相談してるんだよ。ただ……」
「ただ？」
「お姉ちゃんは、バリバリの関数至上主義者なんだよ。普段はPythonとかじゃなくて、HaskellとかSchemeみたいな関数型プログラミング言語を使ってる。関数プログラミングは確かに有効な手法だけど、代数的な思考が身についていない人には難しいと思うんだよ」
「代数って、式の計算だっけ？」
「うん」
「確かに難しいなぁ」
くすくすっ、と笑うイチコ。
「たぶんね、マナブくんは関数型の本当の難しさはわかんないと思う。実を言うと、あたしもよくわかんないとこ、あるんだよ」
「そうなの？」
「うん。関数型は全てを関数で、ということなんだけど、あたしがPythonを使ってるのは、いろんな

方法が使えて便利だと思うからなんだよ。関数型にこだわる必要は、あたしにはわからないんだよ」
「美学かなぁ」
「そうかもね。他にも、オブジェクト至上主義者とか、推論型至上主義者とか……他にも変わったところでは、スタック計算信奉者とか、アセンブラ原理主義者とかいたりするよ」
「パイソニスタみたいなもの？」
「うん、特定言語の信奉者っていう意味では、パイソニスタもそうだよ。もっともパイソニスタは、パラダイム……手法という点では、限りなく中立だけどね」
　イロイロあるらしい……学ぶはそうまとめて深入りしないことにした。
「とはいえ、関数は大切な概念だから、一通りは理解しようね」
「うん」
　ティータイムは終了し、再び画面に向かう二人。
「じゃ、こんどはラムダじゃない方の関数の書き方、いくよ。ラムダは便利だけど、基本的に式しか書けないから、融通が利かないところもあるんだよ。たとえば……三回の入力を受け取って、その値……文字列を','で連結したものを返す関数、っていうのを書くとしたら、ラムダだと面倒だよね」
「えっと……」
「あ、書けないわけじゃないよ」

```
>>> ipt3 = lambda: input('x=')+','+input('y=')+','+input('z=')
>>> print(ipt3())
x=100
y=200
z=300
100,200,300
>>>
```

「ipt3のトコが見づらいよね」
「うん、そのとーり。こういうのは素直に別々に書いた方が見やすいんだよ」
「でも、代入は式にならないんだよね？」
「だから、lambdaじゃなくdefを使うんだよ」
「でふ？」
「defineあるいはdefinitionの略、かな。定義するっていうこと」
「へぇ？」
「こう書くんだよ」

```
>>> def ipt3x():
...     x = input('x=')
...     y = input('y=')
...     z = input('z=')
...     return x + ',' + y + ',' + z
...
>>> print(ipt3x())
x=5
y=6
z=7
5,6,7
>>>
```

「インデント？　ifやwhileのブロックみたいだね」
「うん。良く似てるけど……実は別もの。ifやwhileのブロックは単に実行の区切りだけど、defのブロックはブロックの外と、ある意味隔離されてるんだよ」
「えっと……」
「defブロック内で行った変数への代入は、基本的にブロックの外では無効なんだよ。たとえばこの例ではx,y,zの三つの変数に値を代入してるけど、ブロックの外では、x,y,zという変数は定義されない

し、もしあったとしても変更されない」
「それじゃぁ、外に影響を与えられないってこと？」
「基本的は、ね。外への影響は、最後に書いてあるreturn文で返すことになってる。returnに続く式の値が、関数の戻り値になるんだよ」
「評価されてから返されるんだね」
「そう。もう少し、関数っぽい定義を書いてみるね」

```
>>> def f(x, y, z):
...     return x + y + z
...
>>> f(1, 2, 3)
6
>>>
```

「defの後は関数名、その次が仮引数(かりひきすう)の定義」
「仮引数？」
「引数として受け取ったとき、代入される変数、かな。つまり、def f(x,y,z):としておいて、f(1,2,3)と呼び出せば、xには2が、yには2が、zには3が代入されるんだよ」
「ああ、それで仮引数を使って式や文を書けばいいんだね」
「うん。引数を取らないなら仮引数は書かない。けど、括弧は省略できないから気をつけて」
「呼び出しと同じ形、だね」
「そうだね」
「仮引数は、１文字限定？」
「ううん、基本的に変数だから、変数名として使えるものならなんでもいいよ」
「最後は必ずreturn文で戻り値を書くんだね？」
「省略してもいいよ。return文自体を省略したり、returnだけを書くとreturn Noneの意味になるから」
「でも、戻り値を書かないと、関数の意味、無いんじゃないの？」
「じゃ、こんなの、どう？」

```
>>> def m():
...     print("Hello Manabu!")
...
>>> m()
Hello Manabu!
>>>
```

「あ……」
「外の変数に影響を与えないということは、何もしないってことじゃないよ。print関数だって、コレと同じだよね」
「『外のプログラム』には影響を与えなくても、『プログラムの外』には影響するんだね」
「外のプログラムに影響を与える方法も、じつはいくつかあるんだけど、黒魔術っぽいから、あんまりお勧めできないよ」
「く、黒魔術ぅ？」
「あ、プログラミングで『邪道』くらいの意味。一見便利なんだけど、濫用するとコードが見難(にく)くなって、後で再利用や拡張しようとしたとき、不便になるんだよ。Pythonは、黒魔術を極力排除する言語構造にはなってるけど、それでも『行儀の悪い』コードを『書けない』わけじゃないんだよ」
「ブロックの外のコードに影響を与えるコードは、行儀が悪いの？」
「関数化はブラックボックス化だから、経過はともかくブラックボックス化の中で行われた結果は、できるだけはっきりと判った方がいいんだよ。そうだね、やっぱり例を書いてみようか」

```
>>> def goodF():
...   return 777
...
>>> def badF():
...   global x
...   x = 666
...
>>> x = 100
>>> x
100
>>> x = goodF()
>>> x
777
>>> badF()
>>> x
666
>>>
```

「goodFはそのまま数値を返す関数。badFはglobalっていう特殊な宣言をして、内部変数を外の変数として扱ってる関数。goodFでは呼び出しの際に明示的……目に見える形で代入してるけど、badFでは呼び出した側ではxと何の関係もなさそうなのに、xが変化してるよね。こういうのを変数の暗示的変更っていうんだけど、意味が無い上に、判りにくいよね」
「他人がプログラムの一部だけを見ても、わかんないかも」
「他人だけじゃなくて、自分で書いたコードでも、結構忘れちゃうモノだよ。戻り値以外の影響を**副作用**って言って、画面表示みたいな必要不可欠な副作用もあるけど、変数の暗示的変更みたいな基本的に無意味な副作用は、コードを見づらくするから、できるだけ書かないようにするほうがいいんだよ」
「それが黒魔術？」
「うん。実はこんな例示のための例じゃなくて、global宣言を使うと楽に書ける場面はあるんだけど、それでも何が起こってるか判り難いのは変わらないから、出来るだけ使わないようにしたほうがいいよ」
「うん」
「画面表示とか、ファイルの読み書きとか、副作用を目的とした関数は、本当は関数じゃなくて手続き(procedure プロシージャ)と呼ばれる方が適当だと思うけど、Pythonでは関数という名称でまとめてるんだよ」
「手続き？」
「うん。もともと『プログラムの一連の操作をまとめたもの』という意味。ある言語では、関数という名称の代わりにすべて手続きという言葉を使ってたり、他の言語では関数と手続きを書式で区別しているものもあるよ」
「画面表示まで副作用だとしたら、全部関数、にはならないだろうしなぁ……」
「お姉ちゃんが使ってるHaskelなんかは、難しい手段を使って全部関数にしてるけど、でも、内部を知らない人からみると、副作用を起こしてるようにしか見えないよ」
「結局、不要な副作用を起こす関数は書かない、でいいのかな？」
「うん。できるだけ引数を入口に、出口を戻り値にするようにすること。副作用は必要最低限にとどめること。Pythonで純粋関数型プログラムを書くと、できることが限られちゃうけど、関数型を意識して書くことは、悪くないと思う」
「……ちょっと考えたことあるんだけど、やってみていい？」
「もちろん」
「えっと、前の繰り返しのことなんだけどね……」

```
>>> def f(n):
...     fn = 1
...     while n > 1:
...       fn *= n
...       n -= 1
...     return fn
...
>>> f(10)
3628800
>>>
```

「あ、出来た出来た。階乗計算関数♪」
「……」
「え、えっと……」
「……」
「な、なんか僕、失敗したっけ？」
イチコは寂しそうに笑う。
「マナブくん、優秀すぎ。それ、私が今からやってもらおうと思ってた論題なのに」
「ま、まずかったかな？」
「ううん。マナブくんの成長が速い証拠だから、勿論歓迎すべきだよ。ただ、こうしてマナブくんに教える事って、そんなに長くないのかな、と思うと、ちょっと寂しいかな？」
マナブは笑って言う。
「僕は早く卒業したいな」
「え？」
イチコの表情が変化する前に、マナブは言葉を被せる。
「早くプログラミングが出来るようになって、イチコさんと一緒に何か作れるようになりたい。ううん、多分それは無理でも、イチコさんの手伝いくらい出来るようになりたいよ」
「マナブくん……」
「は、ははは、それまでは、ま、まだまだ前途多難だけどね……って、イチコさんっ！　どーしたのっ！」
全身の血が全て頭に集まったかのように、顔を真っ赤にしたイチコは、そのままふらふらと倒れようとしていた。
マナブは慌てて、その小さな体を支えた。
その体温と、ほんのりと香る少女の匂いに、今度はマナブの意識が飛びそうになるが、なんとか抑えて、イチコを部屋の隅のベッドに運んだ。
ベッドに寝かせた途端、イチコがぱっちりと目を開く。
「……」
「……」
客観的に見れば、マナブがイチコをベッドに押し倒したかのような体勢だ。
そして何を思ったか、イチコがぎゅぅっ、と再び目を閉じる。
「っ！！！」
この極限状態で、マナブの本能は、３つの選択肢を思い浮かべていた。

```
１．なにもしない
２．唇に……
３．下半身に手を伸ばし……
```

イチコが焦れて目を開けようとした瞬間、自分の額に何か柔らかいモノが当てられたのに気づく。
目を開けた時、マナブはイチコに背を向けて椅子に座っていた。
イチコは顔を赤くしたまま、人差し指を唇に当てて、小さく呟いた。
「……中途半端……だよ」

＜後編に続く＞

やｎでレPython
～λの使徒編～

ommentator
In 1945 Peace broke out. It was the end of the Joke. Joke warfare was banned at a special session of the Geneva Convention, and in 1950 the last remaining copy of the joke was laid to rest here in the Berkshire countryside, never to be told again.

1945年、平和が勃発した。これがジョークの最期だった。ジョーク攻撃はジェノバ会議の特殊部会で禁止となった。そして1950年に最後に残されたコピーがバークシアの田舎で眠りにつき、二度と語られることはなかった。

『Monty Python's Flying Circus "The funniest joke in the world"』

■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
・石川 徹(いしかわ とおる):マナブやイチコのクラスメイト
・「λの使徒」:イチコの姉らしいが、不明

第五回「お呼びとあらば即参上　＜後編＞」

　その異変に、周囲はすぐ気づいた。
「お前ら、何かあったのか？」
　トオルは椅子を跨ぐように後ろ向きに座り、マナブに質す。
「えっと……何が？」
「お前と羽生だよ。毎日話してたのに、昨日一日、何の話もしてないって変じゃね？」
「いいじゃないか、そんなこと」
「喧嘩でもしたのか？」
「お前には関係な……」
「いや違うな。この妙な気まずさは……据え膳を食わなかった、とかそんな感じだな」
　マナブの顔が一瞬、キュビズム絵画になる。
「あ、あああ、あ」
「なんだ、図星か。ギャルゲーみたいだな、お前ら」
　慌てふためいて意味不明な言い訳を続けるマナブをしばらく堪能した後、トオルは徐(おもむろ)に切り出す。
「で、実際どうなんだ？　勃たなかったのか？」
　無言。
「いや、そーいうときは、女の方が要らん慰めをして塩を擦込む、ってのが定番だよな。羽生もお前に近づいてないとこ見る限り、誘ってるのをスルーした、とか、そんなだな」
「……ぅるさぃ」
　ほぼ完璧に読まれている。
「しかし、いいのか？」
「何が、だよ」
　不貞腐れるマナブに、トオルは真面目な顔で答える。
「羽生はヤンデレ属性だろ？」
　他人から言われると、妙に腹が立つ。

「あのタイプは、一人にしておくと、妄想を暴走させて、あらぬ方向……大体破滅的な方向に向かうモンじゃないのか？」
　マナブの顔が蒼くなる。
「となると、次に相手と顔を合わせた時、満面の笑みを浮かべていたら間違いなく**デッドエンド**だな」
「あ、ありうる。そ、そろそろ機嫌を取っておいたほうがいいのかな？」
　命に関わるとなれば、途端弱腰になるのが人間である。
「謝り方失敗すると、それはそれで危険だぞ？」
「じゃ、どーすれば」
「確実にデッドエンドから外れる方法はあるけど……」
「是非教えてくれっ！」
「その後は七割がた、Hシーンに突入だぞ？」
「……」
「度胸据えて、やる自信はあるのか？」
　言われて、さらに弱気になるマナブ。
「そ、そういうのは吝(やぶさか)ではないんだけど、も、もうちょっと穏やかに軟着陸する方法は？」
　ヘタレ全開のマナブにトオルは、
「無い」
と、言い放つ。
「大体、エロゲーでハッピーエンドといえば、ヒロインとの濡れ場があるに決まってる。そうでないと、ユーザが許さんっ！」
「エロゲーじゃないっ！」
「それだけ、エロゲーライクの恋愛してて、何を言うっ！」
　はぁはぁ、と、両者息をつく。
「まぁ、言うか言わないかはお前の勝手だが、取り敢えず教えておいてやる」
「うん。ありがと」
「さすがに**リアル殺人**はごめんだからな」
　トオルのアドバイスを聞いたマナブは、真っ赤になる。
「そ、それ……言うの？」
「なんだ。**デッドエンド回避率は100%**だぞ？」
「えっと……そんなこと言ったら、僕らもおしまいだと思うけど？」
「なんで？」
「覿面(てきめん)に嫌われそうな気が」
「ないない。それはありえない」
「そ、そうか？」
「無いぞ。それも、断言してやる。そも、ヤンデレキャラは情の深さが尋常じゃないから厄介なんだ。歪んではいても、基本一途なんだよ。っていうか、そのくらいで嫌う神経なら、苦労無いってば」
「そ、そうかな？」
「ゲームと違って、三角関係とか闖入者とか居ないだろ？　一途には一途を返せば、実は意外に攻略は楽だよ」
「う、うん」
「それとも、既にあのロリバディをしゃぶり尽くして飽きた、とか言うなら話は別だが」
「まだ、殆ど何にもしてないよ」
「ふむ、若干の進展はあった、と」
　また誘爆するマナブ。
「まぁ、羽生と正反対のグラマーな美少女か美女が現れて、お前にちょっかいを出す、なんてお約束展開さえなければ大丈夫だって」
「ギャルゲじゃあるまいし、そんな偶然あってたまるか」

彼は数時間後、そんな偶然に出会うのである。

一人の帰り道。
ついこの間まで、イチコと途中まで帰っていたことが楽しい夢のように思える。
一度得た幸せを失った喪失感は、幸せを得る前には決して想像することはできない。
(決定的に振られる前に、明日はなんとかしよう)
マナブが心中で決意したちょうどその時、前を塞ぐように一人の女性が現れた。
「錦織真武くんね」
不意を打たれ、心臓がバクバクと爆ぜる中、マナブは女性を見た。
身長は、ヒールを履いているとはいえ、決して低くないマナブと同じくらいはある。
すらっとしていながら、凹凸がちゃんと出ている(ファッションモデルではなく)グラビアモデル体型。
栗色の波打つ長い髪、はっきりとした眉の下に、挑発的に笑う鋭い目がある。
美人、と一言で言ってしまえば簡単だが、ある意味魅力のベクトルがイチコと正反対の女性だった。
トオルの与太話が頭に残っていたマナブは反射的に身構えたが、「そんなバカなことがあってたまるか」と自嘲気味に笑い、肩の力を抜いた。
「えっと、あなたは?」
取り敢えずこちらの名前を知っているようなので、聞いてみる。
「そうね、『ラムダノシト』とでも名乗っておこうかしら」
ラムダノシト……『λの使徒』?!
「あ――――――っ!、あなたがあのメールのっ!」
同時に、イチコの科白が甦る。
『多分、あたしのお姉ちゃん』
この女性(じょせい)が、イチコの姉?
あまりの共通点の無さに、マナブの中の常識は、その考えを拒否しようとする。
(いくら、兄弟のパターンは組み替え抜きで64兆通りはあるからっていって、此処まで似てない姉妹って無いだろ)
しかし、ここのところの非常識なイベントを生き抜いてきたマナブは、自分の常識をあまり信用しないようになっていた。
「えっと、イチコさん……羽生一子さんは、あなたがお姉さんじゃないかって疑ってたんですけど……」
「なぁんだ、イチコったら、もう喋っちゃったのね。つまんない」
「へ?」
女性は姿に似合わない子どものような膨れっ面をしたあと、ぺろっと舌を出す。
「じゃ、誤魔化しても無駄ね。じゃ、改めて。私はイチコの姉の羽生 梓(はぶ あずさ)。よろしくねマナブくん」
「よ、よろしく」
差し出された手は華奢で、ひんやりと冷たかった。
その手の感触だけが、イチコと唯一似ていた。
「マナブくん、今から時間ある?」
「え、えっと……」
確かに、イチコのプログラミング教室は休業状態で、マナブは家に帰っても、ゲームくらいしかやることが無かった。
「ちょっとおねーさんに付き合ってくれないかな。今ならイチコの裏情報とか、**特典画像**とか、ついてきちゃうよん」
「行きます」

喫茶店『V-Joke』
この店の名物『Killer Joke パフェ』をつつくアズサに非常な違和感を感じながら、マナブはカ

フェラテを啜っていた。
　入店から10分。
　見た目からは想像できないくらい口の軽いアズサから、イチコの今までのいろんなコトを聞いてきた。
　特典画像とやらは、去年までに捕ったイチコのケータイ写真で、さすがにコレは、というモノを除き、コレクションをごっそりとコピーしてもらった。
　帰ったらCDに焼いて保存しとかなきゃ……
　そして、彼女の生い立ち。
　小学校中学校時代、殆ど友人がいなかったことなどは、本人の口から聞いて知っていたが、家庭のことについては、いままで聞いた事が無かった。
　アズサとイチコがあまり似ていない理由は、簡単にわかった。
　実はアズサとイチコでは、母親が違うのだ。
　アズサの母親は、アズサが６歳の時他界。
　その後アズサの父親と再婚したのがイチコの母親だった。
　家族仲は悪くなく、イチコの母もアズサを可愛がってくれたため、アズサもわだかまりなく腹違いの妹を可愛がることができた。
「イチコって名前は実はね、私がつけたの」
「え？」
　コレは、意外だった。
「お母さん……今のお母さんはね、次女っぽい名前をつけようとしてたみたいだけど、あたしが反対したの。間違いなくあたしの妹だけど、お母さんには最初の子どもなんだから、絶対『一』をつけるべきだって主張したんだよ」
　それで次女なのに『一子』。
「家で可愛がりすぎたのが悪かったのか、家の外では引っ込み思案と不器用が災いして、あまり友達ができなかったみたいね。で、私がプログラム教えたんだけど、かえって人を遠ざけるようになったみたいね」
　あった頃のイチコを思い出して、思わず頬が緩む。
　アズサはにやりと笑い、マナブの額を人差し指で突つく。
「なのに、いきなり友人すっとばして彼氏作るんだもんなぁ」
「済みません」
「謝る事ないわよ。それであの子、以前に比べてすごく積極的になったし。昔から思いつめると周りが見えない性格だったから、恋でもしたら、**ストーカーにでもなる**んじゃないかと心配してたんだけど、うまくいってるみたいね」
　犯罪者(ストーカー)ではなく、あくまで**ヤンデレ少女**の域で収まっている。
「今現在はちょっと難航してますけど」
「そうみたいね。一昨日(おととい)帰ってから、ぼんやりしたり怒ったり、妙な感じだったわよ」
「原因はわかってますが、何があったかは聞かないでください」
「そこまで野暮じゃないから安心して。それに今日の目的は、そもそもイチコじゃないんだから？」
「**関数**、ですか？」
「そうそう」
　そう言って、黒いエナメル革のハンドバッグから、モバイルPCを取り出す。
「Pythonやってるのよね？」
「ええ」
「イチコの趣味とはいえ、良いチョイスね。大学なんかでSchemeが難しくてプログラムできない学生に使ってるみたいだしね」
「プログラミング初心者の入門用には最適だ、とイチコさんは言ってました」
「それは教師次第ね。いくらツールとして優秀でも、旧世代の手続き志向のプログラマにかかれば、あっというまに汚染コードプログラマになれるわよ」
「汚染って……」

アズサはマナブをギロリと睨む。
「いい、マナブくん？　プログラミングっていうのは半ば習慣なの。一旦悪い習慣を身につけたら、改めるのは大変なんだから」
「そうなんですか？」
「そう。例えば再代入。マナブくんは階乗計算をする関数、どう書く？」
「えっと……」
差し出されたモバイルのキーを叩いて、前にイチコに書いてみせたコードを思い出してPythonコンソールに打ち込む。

```
>>> def fact(n):
...   fn = 1
...   while n > 1:
...     fn *= n
...     n  -= 1
...   return fn
...
>>> fact(10)
3628800
>>>
```

「そうよね。ところで、nはいくつ」
「はい？」
意外な質問に、マナブの頭は一瞬真っ白になる。
「nとfnは、whileを回している間、どんどん値が変わってるわ」
「ええ、まぁ、そういうモンかと」
nは回数を数えるカウンタ変数。
fnは経過の記憶用のメモリ変数。
どちらも状態が順次変化していくものだと、イチコは言っていた。
「とすると、nは関数じゃないのね？」
「え？」
目をぱちくりさせるマナブに、アズサは畳み掛けるように言う。
「いい、マナブくん？　学校で習った数学で、問題を解いていく過程で、変更される変数ってあった？」
「えっと……」
マナブが理解するまで、少し間があったが、アズサは慌てなかった。
そしてマナブは、程なく真相にたどり着く。
「そんなの無いですよ。二次方程式なら、解が二つあることはありますけど」
アズサは満足そうに頷く。
「その通り。適用する前ならともかく、一旦式を適用したら、その中で勝手に値が変わったら変よね？」
「え、ええ」
「なのに、この式ではどんどん変化してる。おかしいと思わない」
「え、えっと……」
言われてみれば、そうである。
プログラムを式としてなど見たことがなかったマナブは、改めてその不自然さに気づいた。
「じゃ、whileは使うな、と？」
「そうね、ループはどうしても必要でなければ、使うべきでないと思うわ」
「ループしないとしたら、書けないじゃないですか」
「書けるわよ？」
「え？」
「書いてみよっか？」

```
>>> def fact(n):
...   if n > 1:
...     return fact(n-1) * n
...   else:
...     return n
...
>>> fact(10)
3628800
>>>
```

「あ……」
　確かに、今まで習った関数定義と関数適用、そして条件分岐だけを使って、間違いなく階乗計算が行われている。
「え、あ、あれ？　関数定義の中でその関数使ってる……」
「そういうこともできるのよ。これは**再帰定義**って言って、定義の一部に自分自身を使ってるわ」
「えと、そうすると……定義できないのでは？」
　常識的に考えれば、循環定義になりそうな気がする。
「でも、ちゃんと計算できてるみたいだけど？」
「そ、そうですよね。お、おっかしいなぁ……」
　慌てふためくマナブに、アズサは優しく語りかける。
「この関数がどうやって機能するかを理解するのはちょっと難しいけど、注目して欲しいのは、さっき言った変数の定義よ。関数の中では、一度代入された変数が変更……すなわち**再代入されてない**でしょ？」
「そ、そうですよね。確かに。あ、でも定義の最後でnにn-1が……」
「それは代入じゃなくって、別の関数の呼び出しをしてるだけ。'+'を関数だとすると、(1+2)+3ってやってるのと同じよ」
「'+'が関数？」
「一応、Pythonでは'演算子'って呼んでるけど、本質は同じよ。引数として2変数を取るか、左右の項として受け取るのか、ってだけの違いだもの」
「え……う……あ……」
　マナブの混乱は、頂点にまで達しそうになった。
「さっきも言ったけど、理解しなくてもいいわ。ただ、方法はいろいろあるってこと。そして「プログラムの都合で無理やり」じゃなくって、出来るだけ**自然に**数式を解くようにプログラムするのが、素直で綺麗なプログラミングをするコツなの」
「それが関数プログラミング？」
「そう」
　マナブは、漸く身近に感じてきたプログラミングの深淵の一端を覗き込んだ気がして、背筋が寒くなった。
「幸い、Pythonにはまだまだ豊富なツールがたくさんあるわ。それを全て正しく綺麗に使いこなせば、美しく完全な関数プログラミングの世界が構築されるわ。がんばって」

　その夜、マナブは一睡も出来なかった。

　翌日、登校して席に着いたイチコの元に、マナブがいきなり現れた。
　一瞬、イチコの心臓は高ぶったが、直ぐに据わった目の下に大きな隈を作っているマナブの異様な様子に気づいた。
「ど、どうしたのマナブくん？」
「イチコさん、助けて……」

昨日の微妙なうやむやなど吹っ飛ぶような、すがるようなマナブの声だった。
「そう、お姉ちゃんにそんなこと言われたの」
昨日二人で姉とお茶していたことなど、マナブに言いたいことは山ほどあったのだが、そのアズサの言葉に悩んで一晩中眠れなかったと聞き、流石に今回ばかりは特別に寛大な心になったようだ。
「アズサさんの言う通り、僕自身もコードに対する考え方を変えなきゃ、とか思うのに、そもそもアズサさんの書いたコードの意味、一晩考えてもわかんなかったんだ。僕はもう、どうしたらいいか……」
「簡単だよ。無視すればいいんだよ」
「へ？」
イチコはやれやれ、といった表情で、コンソールを叩く。
「お姉ちゃんの書いたコード、あれ、実は不完全だから」
「ええっ！？」
「見てて」

```
>>> def fact(n):
...   if n > 1:
...     return fact(n-1) * n
...   else:
...     return n
...
>>>
```

「実はこのコード、あまり大きな数値を入れるとエラーを起こすんだよ。やってみようか？」

```
>>> fact(1000)
  File "<stdin>", line 3, in fact
  File "<stdin>", line 3, in fact
            :
            :
            :
  File "<stdin>", line 3, in fact
  File "<stdin>", line 2, in fact
RuntimeError: maximum recursion depth exceeded in cmp
>>>
```

「あれ、1000!って、確か、前に繰り返しで書いた時は通ったよね？」
「そうだよ」

```
>>> def fact(n):
...   fn = 1
...   while n > 1:
...     fn *= n
...     n  -= 1
...   return fn
...
>>> fact(1000)
402387260077093773543702433923003985719374864210714632543799910429938512
...
000000000000000000000000000000000000000000000000000000000000000000000000
>>>
```

「つまり、Pythonのプログラムとしては、マナブくんが書いた方が正しくて、お姉ちゃんのは間違い、ということ」
「どういうこと？」
「その前に、お姉ちゃんの書いた再帰プログラムの動作の原理を説明しておくね」

「うん」
「再帰に限らず、繰り返しになるものは、簡単なものから順を追ってみていくといいんだよ。まず

```
def fact(n):
  if n > 1:
    return fact(n-1) * n
  else:
    return n
```

の構造は、基本的には単純なif-elseの条件分岐だっていうのはわかるね？」
「うん」
「そして、nが1以下の時は、nを返す。これも判るよね？」
「うん」
「さて、あとはnが1より大きい時なんだけど、例えば3としてみようか？」
「う、うん」
なぜ３なのか、と疑問に思ったが、別に深い意味はないと思い返し、マナブは素直に頷く。
「fact(3)の戻り値は、n>1だから、fact(2)*3になるよね？」

```
fact(3) = fact(2) * 3
```

「3-1=2だから、そうだね」
「で、2>1だから、fact(2)はfact(1)*2になる。つまりfact(3)はfact(1)*2*3になるわけだよね？」

```
fact(2) = fact(1) * 2
```

「うんうん」
「さて、1は1より大きいわけではないから、fact(1)は1になる。ＯＫ？」
「とすると……」

```
fact(1) = 1
fact(2) = 1 * 2
fact(3) = 1 * 2 * 3
```

「fact(3)は1*2*3で3!になってるっ！」
「そういうこと」
「そうか、僕は実際に数値を入れずに、関数がどんな値を出すかってことを考えてたから、解けなかったんだ。でも、どうして数が多くなると、できなくなるんだろ？」
「今の話で言うと、fact(3)を答えるには、fact(2)を計算する必要があるよね」
「うん」
「ということは、『fact(3)の答えを出す』っていう操作を、どこかに覚えておかなきゃいけないんだよ」
「そ、そうだよね」
「そして、fact(2)を計算するには、fact(1)を計算する必要があるよね」
「うん」
「そうすると『fact(2)の答えを出す』っていう操作も、どこかに覚えておかなきゃいけないんだよ」
「う、うん」
「この『作業を覚えておく』場所を『作業スタック』って言うんだけど、Pythonではこの作業スタックは1000段弱で制限されてるんだよ」
「えっと……どうして？」
「一つは、際限なくスタックを増やせるようにすると、メモリを使い果たして止まってしまう可能性

があること。もう一つは、スタックトレースっていう実行状態を監視するプログラムを同時に動かしてるからという理由。難しいことはともかく、あまりネストの深い……つまり『関数の中から関数を呼び出す』入れ子があまりに深い再帰は、Pythonの実行時の制限にひっかかるんだよ」
「whileはいいの？」
「whileは一回ずつチェックしてるから、チェックが終わってるのは問題ないんだよ。でも、再帰の場合は前の作業が完了してないから、監視対象のプロセスが膨大にふくれあがっちゃうんだよ」
「うわぁ」
「もちろん、再帰プログラムが有効な場合はあるよ。二分木検索っていうデータベース構造の検索は、再帰でないプログラムを書く方が難しいからね。でも、単純ループに再帰は必要ないよ」
　そう言いつつ、イチコは一つのコードを書く。

```
>>> def fact(n, r=1):
...   if n < 2:
...     return r
...   else:
...     return fact(n - 1, r * n)
...
>>> fact(10)
3628800
>>>
```

「これは**末尾再帰**って言って、次の関数呼び出しの結果がそのままこの関数の結果になるような形にしたものだよ。プログラミング言語や環境によっては、この方法ならスタックを使わない形に最適化してくれるものもあるんだけど、Pythonについていえば、こう書いてもスタックを積むルールになってるから、やっぱり限界はあるんだよ。もっとも、ちょっとだけスタックに積む作業が減るから、メモリの消費は減るけど」
「つまり、必要が無ければ、特に再帰で考えなくてもいいってこと？」
「必要ないよ。それより、お姉ちゃんが余計なことを言って、マナブくんの勉強を邪魔したことのほうが許せない」
　イチコの目が、危険な色を帯びてきたので、マナブは間髪入れず質問する。
「アズサさんは、変数が再代入されるようなプログラミングは『悪い習慣』だって言ってたよ。再代入をしない再帰のほうが自然だって」
　イチコは力強く首を横に振る。
「プログラミングは数学じゃないよ。それから、コンピュータは概念じゃなくて実体だよ。勿論、数学の考え方は便利だから、計算機はできるだけ数学的に振舞うようにしてあるんだけど、プログラムを解くのは結局電子回路だし、動作を考える人間も必ずしも全ての物事を数学的に考えているわけじゃないから、それが全てではないんだよ」
「要するに、全部関数にして考える必要はないってこと？」
「前にそう言った筈だよ。マナブくん、お姉ちゃんに言われて、それ忘れちゃったんでしょ？」
　あっ、とマナブは声を上げた。
　最初に関数を習った時に、イチコは確かにそう言った。

あたしがPythonを使ってるのは、いろんな方法が使えて便利だと思うからなんだよ

「面目ない」
「まさか、グラマーなお姉ちゃんを前にして、舞い上がっちゃったの？」
　冗談めかした言葉の中に、**秘められた殺意**を敏感に嗅ぎ取ったマナブは、何食わぬ顔で言い返す。
「それはないよ。もしドキドキしてて頭が真っ白になるなら、イチコさんと勉強なんて出来ないよ」
　イチコは、自爆して真っ赤になった。
　で、それを気づかないふりをしつつ、マナブはフォローを入れる。
「でも、関数って面白いよね。他に何か、もうちょっと試してみたいな」

「あ、えっと、それなら二次方程式の解の公式とか作って見るのどう？」
「解の公式？」
マナブはメモを取ると、鉛筆でさらさらと書く。

$a \neq 0$ のとき $ax^2+bx+c=0$ の x の解は $\dfrac{-b \pm \sqrt{b^2-4ac}}{2a}$ になる

「ってこれ？」
「そう。マナブくん、数学苦手っていうけど、よく覚えてるじゃない」
「ああ、これは別格。これ暗記しておくと、**因数分解しなくて済むから**」
「それはちょっと……」
イチコはため息をつく。
「今日はプログラミングの勉強をこの話題で切り上げて、因数分解の復習、しよっか」
「藪蛇だぁっ！」
涙目のマナブを慰めるように、イチコが微笑みながら言う。
「でも、ここで解の関数を作っておけば、検算は簡単だよ」
「え？　あ、なるほど」
ぽんっ、と手を叩くマナブ（いいのか？）
「それじゃ、いくつか新しいことを教えておくね。まず累乗(power)の演算子」
「累乗？」
「冪(べき)乗とも言うけど、2の3乗が2×2×2で8とか、そういうの」
「あ、そうだった」
「累乗は乗算に使うアスタリスクの二連続'**'だよ。二つ続けて一つの演算子だから、気をつけてね」
「うん。でも、どうせ二乗までしか出てこないから、掛けてもあんまり変わらないよね。寧ろタイプ数は減るような気が」
ふふふ、とイチコが笑う。
久々に、してやったり、といった楽しそうな顔だ。
「甘いなぁ、マナブくん。確かに2乗はそれでいいけど、多分累乗を使わないと、計算できないよ」
「え？　そ、そう？」
「試しに、Python式で、解の公式の±の＋の方だけでも書いてみて」
「えっと、分母を全部括弧で囲って、(-b+……」
すぐに気づく。
「えっとイチコさん、ルートってどう書くの」
「ほら、ね」
「え？　だ、だってさっきのは累乗でしょ？　へ、平方根なんか関係ないじゃんっ！」
「ところが、そーじゃないんだな。たとえば、 $2^3 \times 2^3$ って、2の何乗？」
「えっと、2の6乗かな？」
「どうやって計算した？　8×8=64を2の何乗って計算しなおした？」
「そんなことしないよ。2を3回掛けたのに、もう一回2を3回掛けたのを掛けるんだから $2\times2\times2\times2\times2\times2=2^6$ ってやったよ。いくら僕が数学が苦手と言っても……」
「つまり、累乗の3について、3+3か3×2をやったんだよね」
「う、うん」
「つまり、 $\left\{\begin{array}{l} x^a \times x^b = x^{a+b} \\ (x^a)^b = x^{a \times b} \end{array}\right\}$ ってことでいいよね？」
「う、うん」
「ところで、 $\sqrt{2}$ ってどんな数値だっけ？」
「2回掛けると、2になる数値のうち、正の方」

「ご名答。ここで提案なんだけど、もし√を累乗で表すとしたら、いくつになると思う？」
「え？」
「ふふ、ちょっと難しいよね。じゃ、式に書いてみよっか
$2=\sqrt{2}\times\sqrt{2}$ で $2=\sqrt{2}^2$ 、ここで、 $\sqrt{2}=2^x$ とすると、 $2=2^{x2}$ で $2=2^{x\times2}$ 、
2は当然 2^1 だから、 $2^1=2^{x\times2}$ よって、 $1=x\times2$ これでどう？」
「え、えっと、 $\frac{1}{2}$ ？」
「ご名答」
「ってことはつまり……あ――――っ！」
急に気づいて書き掛けた自分の式を書き直す。

```
(-b + (b**2 - 4*a*c)**(1/2))/(2 * a)
```

「書けた……」
「面白いでしょ？　ホントは数学用モジュールを導入すればsqrtって関数があるんだけど、別にそんなことしなくても、頭の使い様でなんとでもなることもあるんだよ」
「すごいな、コレ」
「あともう一つはPythonの文法。return文の後にカンマで区切ると、二つ以上の値が返せるんだよ」

```
>>> def f():
...   return 5,10
...
>>> f()
(5, 10)
```

「座標っぽいのが出るね」
「うん。コレはタプルって言って、次に予定してるコレクションっていう、複数の値を一括して扱う方法の一つ。一つずつ取り出す時は、こうするよ」

```
>>> r = f()
>>> r[0]
5
>>> r[1]
10
>>>
```

「[]に囲まれた数字は添え字(index)って言って、先頭が0で、順番に0,1,2,3...と続くよ」
「うん、これで解の関数、書けるね」
「やってみる？」
「うん。取って置きたいから、別ファイルで……って、計算式を使うためのプログラムが要るよね」

```
# sol01.py

def sol(a, b, c):
  r0 = (-b + (b**2 - 4*a*c)**(1/2))/(2 * a)
  r1 = (-b - (b**2 - 4*a*c)**(1/2))/(2 * a)
  return r0, r1

a = 1

while a != 0:
  a = eval(input("a= "))
  b = eval(input("b= "))
```

```
  c = eval(input("c= "))
  print(sol(a,b,c))
```

「こんなとこ、かな？」
「う～ん、バグではないけど、若干動作に問題があるとこあるよ」
「え？　どこどこ？」
「実際動かしてみればわかるよ。a,b,cはそれぞれ簡単なところで1,-7,12とかで動くから」
「うん」

```
C:¥PythonTest>python sol01.py
a= 1
b= -7
c= 12
(4.0, 3.0)
```

「うまく動いてると思うけど？」
「じゃ、終了させてみて」
「うん。a=0は無いから、それをチェックして終了させるようにしたんだけど……

```
a= 0
b= 0
c= 0

Traceback (most recent call last):
  File "C:¥PythonTest>sol01.py", line 14, in <module>
    print(sol(a,b,c))
  File "C:¥PythonTest>sol01.py", line 4, in sol
    r0 = (-b + (b**2 - 4*a*c)**(1/2))/(2 * a)
ZeroDivisionError: float division

C:¥PythonTest>
```

「終わったけど……異常終了だよね、これ」
「そうだよ」
「何でだろ？」
「異常メッセージの内容は？」
「ZeroDivisionErrorって、あ、そうか。チェック入れる前に計算しちゃうから」
「そういうこと。これを回避しようと思うなら……」
「ＯＫＯＫ。皆まで言うな、ってね☆」
「……うん」

```
# sol02.py

def sol(a, b, c):
  r0 = (-b + (b**2 - 4*a*c)**(1/2))/(2 * a)
  r1 = (-b - (b**2 - 4*a*c)**(1/2))/(2 * a)
  return r0, r1

while 1:
  a = eval(input("a= "))
  if a == 0:
    break
  b = eval(input("b= "))
  c = eval(input("c= "))
```

```
  print(sol(a,b,c))

print("The End!")
```

「これで、っと……

```
C:¥PythonTest>python sol02.py
a= 1
b= -7
c= 12
(4.0, 3.0)
a= 0
The End!

C:¥PythonTest>
```

「ＯＫだよね？」
「自分で使うなら、これで十分だと思うよ」
「うんうん」
「でも、改良の余地は一杯」
「うぁ……」
「まず簡単なトコから。手続き／関数は、そもそも繰りかえし使うところを省略したり、メンテナンスしやすくするために書くんだよ」
「メンテナンス？」
「そう。例えば時間が無くて動くだけだった所を高速化するとか、エラーチェックを厳しくするとか。そういう改良をする場合に、操作があちこちに分散してると面倒だよね？　だから関数でまとめて、その関数を変更することで、同じ操作をしてるところ全てが一斉に変更できるんだよ」
「なるほど」
「だから、繰りかえし出てくる記述は、関数でまとめるべきだよね？」
「う、うん」
「ところで、数値入力って、どうなってる？」
「えっと……あっ……」

```
def e_input(p):
  return eval(input(p))

while 1:
  a = e_input("a= ")
  if a == 0:
    break
  b = e_input("b= ")
  c = e_input("c= ")
  print(sol(a,b,c))

print("The End!")
```

「やれやれ」
「次はちょっと難しいよ。sol 関数の中を見ると、r0 と r1 は似たような式を書いてるよね？」
「うん」
「これ、纏められないかな？」
「え？　だ、だって似てるとはいえ、別の式だよ」
「うん。でもできるよ。ヒントは因数分解」
「うきゃっ、に、苦手なんだってばっ！」

「因数分解の基本は？」
「えっと、共役因数のくくり出し……」
「うん。わかったね」
「えっ、えっ……あ、そっか、(b**2 - 4*a*c)**(1/2)は先に計算できるんだ……」

```
def sol(a, b, c):
  rx = (b**2 - 4*a*c)**(1/2)
  r0 = (-b + rx)/(2 * a)
  r1 = (-b - rx)/(2 * a)
  return r0, r1
```

「式自体は三つになっちゃったけど……」
「でも、大分すっきりして見やすくなったよね。もう一段深めると、こんな感じになるんだよ」

```
def sol(a, b, c):
  rx = (b**2 - 4*a*c)**(1/2)
  rf = lambda x:(-b + x)/(2 * a)
  return rf(rx), rf(-rx)
```

「え？ **関数の中に関数書いていい**の？」
　この反応を聞いて、イチコの表情が少し曇る。
「いけないなんて、今まで一言も言ってないよ。それに**マナブくん、書こうとも思わなかった**でしょ？」
「え、う、うん」
「ダメだよ、そーいうルールに転換するような言い方しちゃ。そんなことしてると、お姉ちゃんの言う『悪いプログラマ』になっちゃうよ」
「アズサさん？」
　イチコはこっくりと頷く。
「お姉ちゃんの関数偏重は正直どうかとは思うけど、関数型プログラミングに対する偏見は確かにあるよ。『ビジネスで使うプログラムに、数字遊びは要らない』とかね。関数型の概念が理解できないだけなのに、あたかもそれを『邪道』として見下して、出来ないことを正当化しようとする人なんていくらでもいるんだよ」
「そう言われれば、負け惜しみだったかも。ごめんなさい」
「うんうん。素直に認めるマナブくんは、将来ウィザード間違いなしだよ」
「ベーダー卿みたいに、ダークフォースに目覚めたりして」
「ダークフォースは滅びるんだよ」
「闇の力は魅力なんだけどなぁ」
「中二病みたいなコト言ってると、小難しい儀式ばっかり多い言語や、裏技だけ充実した言語、果てには機械語マニアとかになるよ？」
「僕は基本的にマニアック傾向なんだけどなぁ、理系とか貧乳とかちっちゃい娘とかどじっ娘とか……」
　イチコは無言でマナブを叩(はた)いた。

【内容】
第４回
・lambda 式(無名関数／関数リテラル)
・def 文(関数定義)
・引数／仮引数
・return 文

第５回
・再帰定義
・末尾再帰
・スタックオーバーフロー
・'**'演算子(累乗演算子)
・プライベート関数

【注釈】
第４回
スクリブラー(Scribbler)：この名前、実は『三文文士』とか『へぼ文士』とか言う意味。
冒頭の Python スケッチ：記念すべき第一回目放送の、さらに有名な通称"Killer Joke"スケッチの冒頭。
お呼びとあらば即参上：コズモレンジャーのアレ。「J9 って、知ってるかい？」
『λ の使徒』：『λ 騎士団』というのはあるらしいのだが……
関数(型)プログラミング(functional programming)：「機能的なプログラミング」と読めないこともないが、一応、関数を基本とするプログラミング手法。関数の定義は作中で出てくるが、引数とただ一つの値が対応していること（選択透過性もこの一種）と、副作用の排除が基本。
手続き型プログラミング(procedural programming)：プログラミングを手続き(procedure)に分割してコンパクトにまとめるプログラミング手法。旧世代の手法として、オブジェクト指向プログラミングや関数プログラミングや推論型プログラミングの対極としてよく借り出される言語だが、本来は非構造化プログラミング言語に対する『革新的な』構造化プログラミング言語についての名称。
関数(function)：数値に対応する写像。要するに、ある数値が決まれば、自動的にある値が決まるという対応。その対応を式で書いてもいいが、全ての対応を列挙してもそれは実は関数。基準の数値も対応する数値も、連続値である必要はない。
きょにゅう(巨乳)：大きい乳房のこと。サルは尻でセックスアピールをするが、人間は二足歩行のため、顔の前に尻がこないので、乳房が膨れてセックスアピールするようになった、という説があるらしい。ちなみに巨乳好き→マザコン、貧乳好き→ロリコンというようなものすごく簡単な仕分けをする人もいるようだが、そんなら幼児は全員、欲情しながら母親の乳房を咥えているのか？　という、当然の疑問にぶち当たる（乳離れしないのは未成熟だが、セックスアピールに反応するのは成熟した証拠）。ちなみに、乳腺からの母乳の出は乳房の大きさには全く関係無い。というかボリュームはほぼ脂肪。
　なお『虚乳』という字を当てると、全く逆の意味になる。
あさいんめんと(assignment)：割り当て。普通『代入』と訳す。
つき合ってなんかないっ！：これを口走らないと、始まらないラブコメはやたら多い。
デッドエンド(dead end)：本来の意味は「袋小路」。でもこの場合は素直に「死んでお終い」
ラムダ(λ)：ギリシアアルファベットのＬ音に相当する文字。
フェチ：「萌え」という言葉の、最も近い訳語。
Σ とか Δ とか：Σ は合計、Δ は変化量によく使う。しかし考えてみれば、理系にそんなに詳しく無い高校１年生が、よく知ってるよなぁ。
λ 式／λ 計算：「式の演算結果の値」ではなく「式そのもの」を扱う計算。なお、作中に出てくる以上の λ 式に関する情報は、Lisp の本か、Haskell の本あたりを読むとイヤという程出てくる。
写像(map)：対応。プログラミング言語では、辞書や連想配列やハッシュを指すこともある。

input 関数：引数に文字列を取り、この文字列をプロンプトとして、キー入力の結果を文字列として返す関数（ver.2.x 以前の raw_input 関数に相当）Python は全部「関数(function)」主義だから仕方ないので関数というが、どこをどう見ても手続き(procedure)。
引数(ひきすう argument)：呼び出された関数やコマンドに渡される値。関数が実体化するのに必要な値。引数を取らない関数というのもあるのだが、本来的な「関数」の定義からすれば、対応の無い数値なので、単なる定数になる筈である。
戻り値(return value)：呼び出された関数が戻す値。「返り値」と訳す場合もあるが、漏れなく誤変換で「返り血」になる。関数的な数式の記述を許しているプログラミング言語では、引数を与えて実体化した関数と入れ替わることのできる値。
len 関数：lengt(長さ)の省略形と思われる。引数にコレクションを取り、コレクションの要素数を返す関数。
None：『空』を表す特殊オブジェクト。戻り値や return 文を省略した関数が戻す値のデフォルト。真偽判断では偽と判定される。以前はただの登録オブジェクトだったが、現在では変更不能な予約語扱いとなっている。
「λ 式で書いてみようか」：一般的に、伝統的な言語で入門したプログラマは、手続き的に考えるため、def による関数定義から始めるが、まがいなりにも数学を習っている場合は、lambda のほうがはるかに直感的ではなかろうか。
文系／理系：一説に「利巧系」と「分系」という分け方で、水平分割ではなく上下に分割するための分け方だったと聞くが、私の感覚としては、人文主義(ヒューマニズム)に立脚する考え方と、客観自然史に基づく考えの分類に落ち着いたような気がする。どちらにせよ、学問レベルの話で止めておいていただきたい（世界や現実といったものを、哲学をすっ飛ばして話そうとすると無理がある）
関数至上主義者：ほぼ「Lisper」と同義語。現在でこそ「Lisp は関数型としては甘い」などという原理主義者も見受けられるが、太古の昔から関数型に先鞭をつけたのは Lisp と言って間違い無いだろう（FORTRAN も数式ではあるが）。
オブジェクト至上主義者：登場は 70 年代というオブジェクト指向プログラミング言語だが、90 年代から脚光を浴び、現在下火……というか、安定中。現在の言語では基本的にオブジェクト方式を採用しているものが多いので、基本概念の一つになりつつある。オブジェクトというか、メッセージ指向にこだわるのは、Smalltalker だけか。面白い話だが、抽象データ型⇒オブジェクトは関数の発展形とオブジェクト指向の立場は考えているようだが、副作用を基本とするオブジェクト操作は、関数型からすれば「旧式の手続き型の同類」と思われているらしい。データを主体とするか操作を概念化するかという両極端のため、両者の溝は想像以上に深いと思われる。よって、日和った Python でコードを書くと、オブジェクト型か関数型かで、全然違うタイプのコードが書けてしまう。
推論型至上主義者：推論型は、Prolog とともに長い間一部の人々以外には知られない特殊な分野だったが、最近では関数型の必須要素として、関数型言語と融合する形で復活しつつある。推論型のプログラミングは、手続き型の延長には無いため、プログラミング概念を根本から学習しなおす必要がある。
スタック計算信奉者：RPN（逆ポーランド記法）による後置法がとっても好きな人々。
アセンブラ原理主義：日本人に、かなり多く見受けられる（機械語／アセンブり言語関連の書籍の充実は異常）。もっともコードハッカーがかなり混じっているように思われる。
def ブロック：if や while のような文脈ブロックと、def や class のような定義ブロックは、書式が似ているので混同しやすいが、全くの別物。if や while にはスコープルールが適用されないが、def や class には、それぞれ独自のスコープルールが適用される。このブロックの概念がやや複雑なのが、Python の初心者入門の実質上の難関になっているかもしれない。
仮引数：関数定義内で使用できるシンボル。Python では、シンボルは関数定義内やクラスメソッド内でしか使用できない。関数定義内で使用できるシンボルは、仮引数の他には自由変数（スコープの外を参照した値）があるが、特殊オブジェクト(None)や組み込み関数以外のオブジェクトを、自由変数として参照するのは、あまり望ましくないため、必要な値は引数→仮引数と渡すのが原則である。

黒魔術(black art)：「行儀の悪いプログラム」とも言われる。手軽なのだが、長く使い続けたり改良したりする可能性のあるプログラムに使用すると、自分あるいはメンテナンスする人間に、軽い苛立ち～自殺などの影響を及ぼすプログラミングスタイル。判りづらい副作用などがその例。
３．下半身に手を伸ばし……：「乱れたスカートを直してやる」と続くのですが、何を想像しましたか？

第５回
冒頭のスケッチ：第四話と同じく、"Killer Joke"の、最後の方の引用。
キュビズム：ピカソのアレ。
ギャルゲー：恋愛シミュレーションゲームのこと。パラメータ重視だった昔は、イベントに入るまでひたすらパラメータアップの作業を行う、恋愛とは似ても似つかないゲームだったが、現在ではマルチストーリータイプのものが多くなり、ちゃんと擬似恋愛を楽しめるようになったらしい。ちなみに、今の特殊効果バリバリのものでなければ、プログラミング自体は非常に簡単なので、プログラムを始めようとする人には最適。ただし内容の出来はプログラムではなくスクリプト（脚本という意味と、システムという意味が混在する非常に特殊な例）に依存するので、あまり期待しないほうがいい。
ヤンデレ属性：思うのだが、イチコ程度でヤンデレと言うのは、看板に偽りありなのではなかろうか？　もっと血飛沫が飛び散るようなホラーな展開を考えたほうがいいのかもしれない。
エロゲー：成人指定恋愛及び性的行為シミュレーションゲーム。この業界に疎い方が勘違いされるのは、「美少女ゲーム」と言った場合、通常は前出のギャルゲーではなくこのエロゲーを指す。プレイしたことの無い人、およびプレイしている場面に出くわしたことのない人は、にたにたとイヤらしい薄笑いを浮かべながら股間をまさぐりつつマウスをクリックする怖いプレイ風景を想像するのだろうが、そういうのはエロゲーの中の「抜きゲー」と呼ばれる実用一点張りのモノをプレイしている時に限られる。実際にはフラグやルート解析のため、ノートを右手に、画面を睨みつけるようにプレイしている光景が一般的。濡れ場が入ってたり、ヌードやファックシーンを克明に描写しているから成人指定になっているだけで、本質的にはギャルゲーと変わりないものも多い上、そういったイベントを削除して年齢制限解除版になるものも多い。この場合、ポルノ的グラフィックは、週刊誌のヌードグラビア的な役割程度しか果たしていないことになる。
ファッションモデル／グラビアモデル：ファッションモデルは服装が主役なので、手足のすらっと長いスレンダーな人が多いが、グラビアは普通肉体が主役なので、肉感的なモデルが多い。一口にモデルといっても色々あるので、「モデル体型」とかいう言葉には殆ど意味が無い。
梓：梓巫女という言葉があるとおり、やっぱり神道／巫女関連の言葉。となると、三女(イチコやアズサの妹)の名前は……。ちなみに私を知る人は某ゲームキャラから取ったと思われるかもしれないが、全く関係無いです（ちなみに、某ゲームって、アイマスじゃないよ）
V-Joke："Killer Joke"スケッチに出てくる、イギリスのジョーク爆弾に対抗してドイツが「報復」として放ったジョークミサイル。元ネタ(何でVなのか、とか)は誰でも知ってるので言わないが、とりあえずV-Jokeは、とっても寒いジョークらしい。
特典画像：イチコの水着姿とか、寝姿とかあったらしい。マナブがどっちに興奮したかは内緒。ちなみに「さすがにコレは」というのは、溝に落ちて泣いている幼い頃のイチコの写真だそうな（何だと思った？）
汚染コードプログラマ：具体的にどういうプログラマを指した言葉なのかは謎。ただし、プロのプログラマであれば、中身はともかくユーザに迷惑をかけるようなプログラムだけは避けて欲しい
カウンタ変数：ループ回数とともに、一定に増えつづけたり減りつづけたりする変数。再代入の例としてよく槍玉に上がるが、カウンタ変数ごときがコードを汚染する可能性などあるのだろうか？
メモリ変数：一時的に値を記憶しておくための変数。一時変数とかメモ変数とか呼ばれることもある。
nは関数じゃない：普通は変数を関数とは見ない。
再帰定義：関数定義の中で、自分自身を呼び出す定義。単純ループで再帰を使うのはマニアだけだが、ニ分木検索など、再帰定義で書かないととてつもなく面倒になるプログラムは沢山ある。
循環定義：定義において、定義すべき概念が、これとほぼ同義の語によって定義されるもので、定義の形式はとっているが、表現上の言い換えにすぎないもの[広辞苑　第五版]

'+'を関数：Lisp では、演算子と関数の区別が無いが、Python では関数はオブジェクト(オペラント)、演算子はオペレータとして、純然たる区別がある。
末尾再帰：関数の戻り値が、再帰で呼び出した関数の戻り値と一致するタイプの再帰定義。理論的にはスタックに積む必要は無いので、末尾再帰を単純ループに最適化する言語もあるが、Python の作者の Guido は、スタックトレースがしにくくなるという理由で、末尾再帰の最適化には公然と反対している。実際、最適化した結果ループになるのなら。なぜ最初からループを書かないのか、そちらのほうが気になる。
二次方程式の解の公式： $ax^2+bx+c=0$ で $a\neq 0$ ならば、

$$(ax^2+bx)+c=0$$
$$a\left(x^2+\frac{b}{a}x\right)=-c$$ ここで $\left(x^2+\frac{b}{a}x\right)$ について平方完成を行う
$$\left(x^2+\frac{b}{a}x\right)=\frac{-c}{a}$$

※平方完成： $(x+a)^2=x^{+2ax}+a^2$ より、 $x^2+a=\left(x+\frac{a}{2}\right)^2-\left(\frac{a}{2}\right)^2$

$$\left(x+\frac{b}{2a}\right)^2-\frac{b^2}{4a^2}=\frac{-c}{a} \qquad \left(x+\frac{b}{2a}\right)^2=\frac{b^2-4ac}{4a^2}$$
$$\left(x+\frac{b}{2a}\right)^2=\frac{b^2}{4a^2}+\frac{-c}{a} \quad =====> \quad x+\frac{b}{2a}=\pm\sqrt{\frac{b^2-4ac}{4a^2}}$$
$$\left(x+\frac{b}{2a}\right)^2=\frac{b^2}{4a^2}+\frac{-4ac}{4a^2} \qquad x+\frac{b}{2a}=\frac{\pm\sqrt{b^2-4ac}}{\pm\sqrt{4a^2}}$$

$\frac{\pm x}{\pm y}$ は $\frac{x}{y},\frac{-x}{y},\frac{x}{-y}=\frac{-x}{y},\frac{-x}{-y}=\frac{x}{y}$ で、結局 $\frac{\pm x}{y}$ であるので、

$$x+\frac{b}{2a}=\frac{\pm\sqrt{b^2-4ac}}{2a}$$
$$x=\frac{-b}{2a}\pm\frac{\sqrt{b^2-4ac}}{2a}$$ という公式
$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

因数分解しなくて済む：実話。作者は因数分解が理解できなかったため、解の公式を濫用した経験がある。結果、数学が全体的に出来なくなってしまうというとんでもないしっぺ返しを喰らう。数学の先生にお願いなのだが、出来ない生徒には、沢山問題を解かせるのではなく、まずゆっくり一つずつ説明して、理解できるようにしてあげて欲しい。
累乗／冪(べき)乗：整数の累乗は誰でも覚えているが、分数(小数)にすると、累乗根になるということは、意外に忘れられる（あるいは知らないか）。Python には math.sqrt という関数があるが、モジュールをわざわざインポートせずとも、累乗演算子を使えば書ける。ちなみに、Python の ver.2.x 以前を使っている場合は、'**(1/2)'とすると 1/2 が 1 になるので、'**(1.0/2)'か'**0.5'を使うと良い(三乗根なら'**(1.0/3)')。なお虚数の計算をしようとして(-1)**0.5 とかすると、

```
>>> (-1)**0.5
(6.123031769111886e-17+1j)
>>>
```

のように、妙にミクロな誤差が出てくるので、桁あふれを利用して……

```
>>> (-1)**0.5+1-1
1j
>>>
```

と、消しておくと綺麗に出る。

タプル：Python のコレクションオブジェクトの一つ。要素が変更不可能なリスト。要素をカンマで区切れば自動的に書けるため、拡張文法の役割も果たす隠れた立役者の一人である（もう一人は辞書）

バグではないけど：製品ならば明らかにバグだが、自分で使う分には問題ない、という意味。使い捨てスクリプトならば、子の程度で十分。

if a == 0: 0 は偽なので、if not a:というような書き方も可能。ちなみに、マナブは初心者なので、イチコに習った通り break 文をブロックで書いているが、普通は一行で書いてしまうのが普通。

数値入力：ver.2.x では input 関数で入力をそのまま評価することが出来たが、eval 関数など、評価をともなう入力はプログラムの安全上問題が多いので、手軽に使える raw_input 関数と差し替えた形になる。作中の e_input 関数は、単純に以前の input 関数と同じ形にもどしただけである。数値を確実に入力するように監視するには、以下のような方法が考えられる。

```
def e_input(p):
  while 1:
    try:
      s = input(p)
      for x in s:
        if not x in "0123456789-": raise Exception('input error')
      r = eval(s)
      break
    except Exception as E:
      print(E)
      print("input number!")
      continue
  return r
```

関数の中に関数：ここで使用されているのが lambda 式であり、lambda 式は定義というより『関数リテラル』とみなしたほうが判りやすいので、マナブの主張は色々な意味でズレている。なお、自分が思いつかなかったことの悔し紛れにルールを云々言うのは、まさに「コロンブスの卵」。とはいえ、無意識によくやってしまう誤りであり、さらに防衛のために意地になってその後もその方式を否定してしまうこともザラにある。幾分作者の自戒をこめて書いた。

関数型プログラミングに対する偏見：「遊びじゃないから」という言葉は、実際にプロから聞いたことがある。その他、関数型ではないが「フリーウェアのスクリプトは、仕事には危なくて使えない」という言葉も。商用とフリーソフトウェアのどっちが危険なんだか……

ダークフォース：Pythonista にとっては、Perler は帝国軍らしい。

中二病：男子が中学二年生時に取りがちな痛い行動・思想（『中二病取扱説明書』新紀元社）

理系とか貧乳とかちっちゃい娘とかどじっ娘とか……：いつの間にか『理系』も、萌え対象（要するに欠点）になっているらしい。

＜マナブの書いた最終的なコード＞

```
# sol03.py

def sol(a, b, c):
  xf = lambda k:(-b+k*(b*b-4*a*c)**0.5)/(2*a)
  return xf(1),xf(-1)

def e_input(p):
  return eval(input(p))

while 1:
  print("x = ax^2 + bx + c")
  a = e_input("a= ")
  if a == 0: break
  b = e_input("b= ")
  c = e_input("c= ")
  print("x -> ", sol(a,b,c))

print("The End!")
```

＜おまけ＞１行で書く解の公式

```
>>> sol = lambda a,b,c:[(-b+k*(b*b-4*a*c)**0.5)/(2*a) for k in (1,-1)]
>>> sol(1,-7,12)
[4.0, 3.0]
>>>
```

【Python あれこれ】３．lambda 式の制限と楽しみ

他言語を学んだ人が Python で違和感を覚える点は(決して多くないとはいえ)幾つかあるようです。その中でも、lambda 式の内容が一つの式で表されなければならないという制限は、書式の流儀とかいった瑣末な違いではないため、ある程度 Python を評価してくれる方でさえ、首をかしげることが多いようです。一応 Guido としては lambda は「コールバック関数などの小さな無名関数専用」なのであり、大規模な手続き的な関数は def を使うよう、勧められていますが、今まで文だった exec が関数になった Python3 以降では、実際には制限は有って無きが如しとなっています。

もっとも私は、lambda の制限が逆に「式でモノを考える」訓練に最適だと考えています。実行順序を制御するのが難しく(if-else 式を悪用すれば可能ですが)、操作の流れ(フロー)で書きづらい lambda 式であればこそ「最終的にどんな値を出せばいいのか、ということを突き詰めてロジックを組み立てる必要があります。これはハマると市販のパズル並に楽しめます。時間の無い時には手続きで書いて、時間があるときには lambda でアリゴリズムを整理して、一粒で二度(実用/趣味)美味しいスグレモノです。

また、作中では初心者に最初に def 定義ではなく lambda 式から始めていますが、これは用語解説にも書いた通り、関数的なものは lambda のほうが理解しやすい、と思ったからです。個人的には「手続きは def、関数は lambda」の使い分けがベストかな？　と思うのですが、いかがでしょう？

（機械伯爵）

（投稿日付: 2009 年 8 月 6 日）

神経衰弱

written by Yさ

1. このゲームは

前作(STATUS=いわゆる文字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六弾。
今回は、カード配置を高度な記憶力と勘で推理するゲームです。

前作STATUSでストックが尽きたので、今回は新たにトランプゲームを作ってみました。
...要するに他に適当なネタが思いつかなかった＝やっつけ、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 memory.awk[Enter]
~~~~~~~~~~~~~~~~~~~~~~~~~~
```

等で起動するとゲームスタートです。

1) 場に7×7枚のカードがランダムに伏せて置かれます。またランダムな3枚が場から除外されます。
めくるカードを1枚づつ、縦a～g,横1～7の文字の組み合わせで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。

例) a1[Enter]

指定した2枚がペアの場合＝カードのランクが一致していればそのカードを獲得し、さらにもう2枚めくることができます。ペアをめくり続けている間は繰り返しとなります。

2) ペアとならなかった場合は、カードが再び伏せられ、コンピュータ側の番になります。

3) 以後、交互に、場にペアとなるカードが無くなるまでカードをめくっていきます。
場に1枚(または3枚)残った状態でゲーム終了となります。
ペアの獲得数の多い方が勝者です...コンピュータに勝つためにはかなり高度な記憶力と勘が必要です(^^;

[蛇足]
スクリプト内 function cardDisp() にもコメントしていますが、常にコマンドプロンプトのクリアをしてから場を表示するようにしています。
環境によっては "cls" ではダメなので、 "clear" をお試しください。

4. 開発環境など

ソースは、今回 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。
題材は一番簡単と思われるトランプゲームを選んだつもりだったのですが、例えば、コンピュータにどうやってカードを選ばせるか等、いざプログラム化すると結構手間取りました。
「既にめくったカード」「場から無くなったカード」の枚数の表示は元々動作確認用だったのですが、人間向けのヒントとしてそのまま残すことにしました(^^;
後、コンピュータの戦略として'適当に間違える'接待モードがあった方が良かったかもしれませんね...

5. その他

本プログラムはフリーソフトです。
著作権は作者であるＹさにあります。
ただし転載、再配布、改造、消去は自由です。
(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。

6. スクリプト

スクリプトを以下に示します。

[memory.awk]

```
##  MEMORY  written by Ｙさ

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

BEGIN{
    ##  カード
    split("H,D,C", StrSuit, ",");  StrSuit[0]="S";
    split(" 2, 3, 4, 5, 6, 7, 8, 9,10, J, Q, K", StrRank, ",");  StrRank[0]=" A";
    ## Cards[key, 52];       ##  カード本体
    ## Table[52];            ##  台札

    split("b,c,d,e,f,g", NtoA, ",");  NtoA[0]="a";

    srand();       ##  乱数の初期化
    makeCards();   ##  カードを用意して...
    shuffle();     ##  カードを切り...
    ## 場でオープンした札のリストを準備する
    ## Peep[13],Take[13];    ##  登場済み,獲得済み数字 A～K=0～12
    ## CardStat[52];         ##  0=未オープン, 1=オープン済, 2=指定, 4=獲得済み
    for(i=0; i<52; ++i) CardStat[i]=0;
```

```
    ## Turn;       ##  順番 0:人→1:COM
    Turn = 0;
    ## sc[2];      ##  獲得回数 0=人,1=COM
    sc[0]=sc[1]=0;

    ##  場にペアが残っている間繰り返し
    GetMax=48/2 - (( ! (isPair(49,50) || isPair(49,51) || isPair(50,51)))?1:0);
    while(GetMax>sc[0]+sc[1]){
      cardDisp(0);
      if(Turn==0) man(); else com();
      if(++Turn>1) Turn = 0;
    }

    ##  Game Over
    cardDisp(1);
    exit;
}


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

            Table[p] = p;
        }
    }
}


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


##
```

```
## ペアかどうか
##
function isPair(p1,p2){ return Cards["rank", Table[p1]] == Cards["rank", Table[p2]]; }


##
## 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<=7){
    s=tolower(substr(str,1,1));
    for(c=0; c<7; ++c) if(s==NtoA[c]) return (r-1)*7+c;
  }
  return -1;
}


##
## 場の表示
##
function cardDisp(endSw,  r,c,str,pos)
{
  system("cls");  # clear screen--if cls does not work, try "clear"

  ## 0....5....0....5....0....5....0....5....0
  ##     -a-  -b-  -c-  -d-  -e-  -f-  -g-
  ## 1) [%%%][%%%][%%%][%%%][%%%][%%%][%%%] (1
  ##    [???][???][???]     COM:99 / 99:YOU
  print "    -a-  -b-  -c-  -d-  -e-  -f-  -g-"
  for(r=0; r<7; ++r){
    printf("%d) ", r+1);
    for(c=0; c<7; ++c) printf("%5s", cardString(endSw, r*7+c));
    printf(" (%d¥n", r+1);
  }
  print "    -a-  -b-  -c-  -d-  -e-  -f-  -g-"
  printf("¥n   ");
  for(pos=0; pos<3; ++pos)
    printf("%s", (endSw!=1)?("[???]"):cardString(endSw,7*7+pos));
  printf("     COM:%-2d / %2d:YOU¥n¥n", sc[1], sc[0]);

  printf("Peep:");  delete Peep;  dispStat(Peep,1);
  printf("Take:");  delete Take;  dispStat(Take,4);
  printf("%*s%s", 5, "", StrRank[0]);
  for(r=1; r<13; ++r)  printf("|%s", StrRank[r]);
  print "¥n";
}
function cardString(endSw,pos)
```

```
{
  if(endSw!=1 && (CardStat[pos]==0 || CardStat[pos]==1))
    return "[%%%]";
  if(CardStat[pos]==2 || CardStat[pos]==3 || endSw==1 && CardStat[pos]!=4){
    if(endSw==1) CardStat[pos]=1;
    return sprintf((endSw==1)?("[%s%s]"):("<%s%s>"),
      StrSuit[Cards["suit", Table[pos]]],
      StrRank[Cards["rank", Table[pos]]])
  }
  return "";
}
function dispStat(tbl,stat,  i,r)
{
  for(i=0; i<49; ++i)
   if(CardStat[i]==stat || CardStat[i]==stat+2) ++tbl[Cards["rank", Table[i]]];

  printf("%2d", tbl[0]);
  for(r=1; r<13; ++r) printf("|%2d", tbl[r]);
  print "";
}


##
##  人間main
##
function man(  tmp,p1,p2)
{
  ##  2枚のカードの位置を入力
  p1=which();  CardStat[p1]+=2;  cardDisp(0)
  do{ p2=which(); }while(p1==p2);
  doGet(p1,p2);
}
function which(  p,tmp)
{
  do{
    printf("Which?> ");
    p=-1;  do{ tmp="";  getline tmp; }while(tmp=="");  p=toPos(tmp);
    if(p<0) p=toPos(substr(tmp,2,1) substr(tmp,1,1));  ## (念のため)1,2文字目を入替
  }while(p<0 || (p>=0 && CardStat[p]==4));
  return p;
}


##
## 獲得判定
##
function doGet(p1,p2)
{
  if(CardStat[p1]<2) CardStat[p1]+=2;
```

```
  if(CardStat[p2]<2) CardStat[p2]+=2;
  cardDisp(0)
  if(Turn==1)
    printf("Com try> %s  %s%d %s%d¥n",
      Strategy, NtoA[p1%7], int(p1/7)+1, NtoA[p2%7], int(p2/7)+1);
  if(isPair(p1,p2)){
    CardStat[p1]=CardStat[p2]=4;
    ++sc[Turn];
    --Turn;  ## もう一度めくる事ができる
  }else{
    CardStat[p1]=CardStat[p2]=1;
    print "... no match";
  }
  printf("(push Enter)"); getline tmp;
}


##
##  COM main
##
function com(  i,n,p,x,list,cnt,q)
{
  ## オープン済みのペアを探す
  Strategy="'well-known'";
  x=0;
  delete p;
  for(n=0; n<13; ++n){
    if(Peep[n]>=2){
      for(i=0; i<49; ++i)
       if(CardStat[i]==1 && Cards["rank", Table[i]]==n) p[x++]=i;
      doGet(p[0],p[1]);
      return;
    }
  }

  ## 未オープンの中から１つ選び...
  Strategy="'previously-existing 2nd'";
  cnt=0;
  delete list;
  for(i=0; i<49; ++i) if(CardStat[i]==0) list[cnt++]=i;
  if(Peep[n=Cards["rank", Table[p[0]=list[q=rnd(cnt)]]]]>=1){
    ## オープン済みの中からペアを探す
    for(i=0; i<49; ++i){
      if(CardStat[i]==1 && Cards["rank", Table[i]]==n){
        doGet(p[0],i);
        return;
      }
    }
  }
```

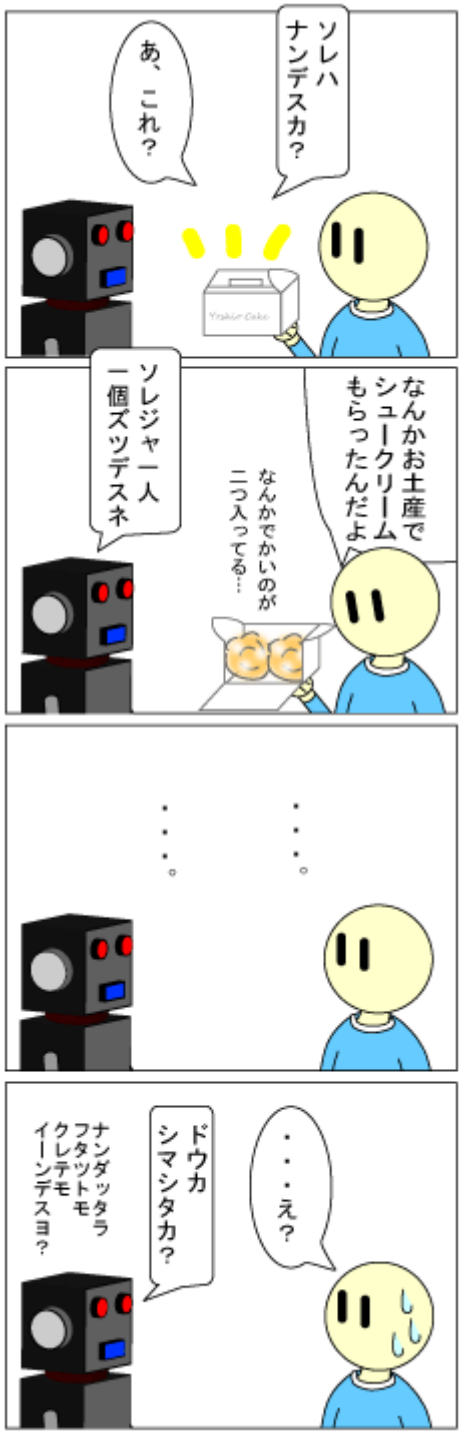
```
  ## 未オープンの中からもう1つ選ぶ
  Strategy="'at random'";
  list[q]=list[cnt-1];  --cnt;
  doGet(p[0],list[rnd(cnt)]);
}
```

(投稿日付: 2009年8月12日)

よしおさんとロボ太

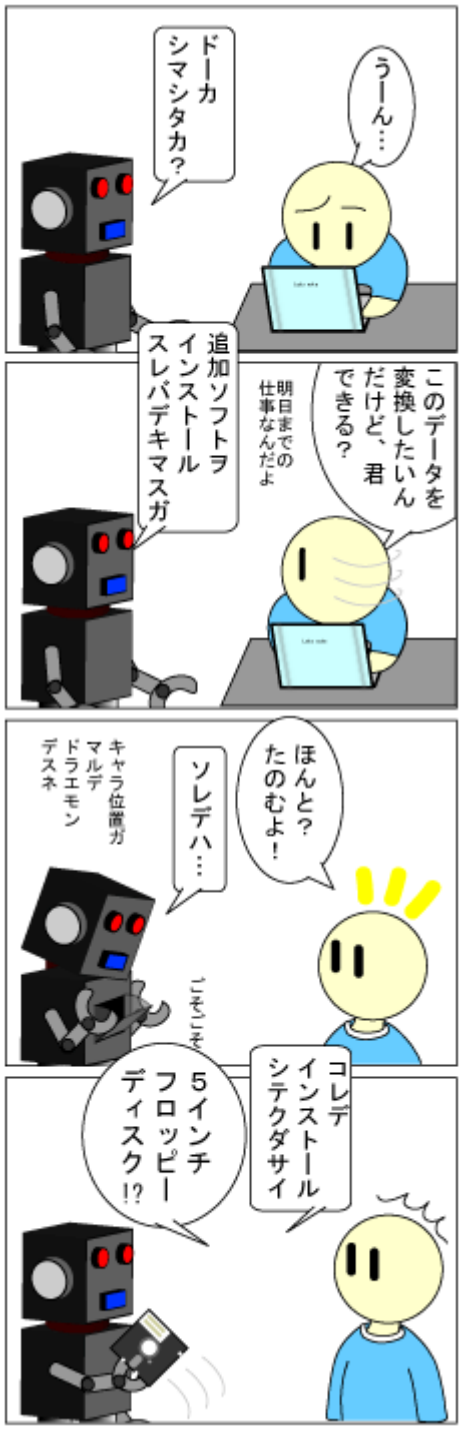
海鳥作

「続・ロボ動力」



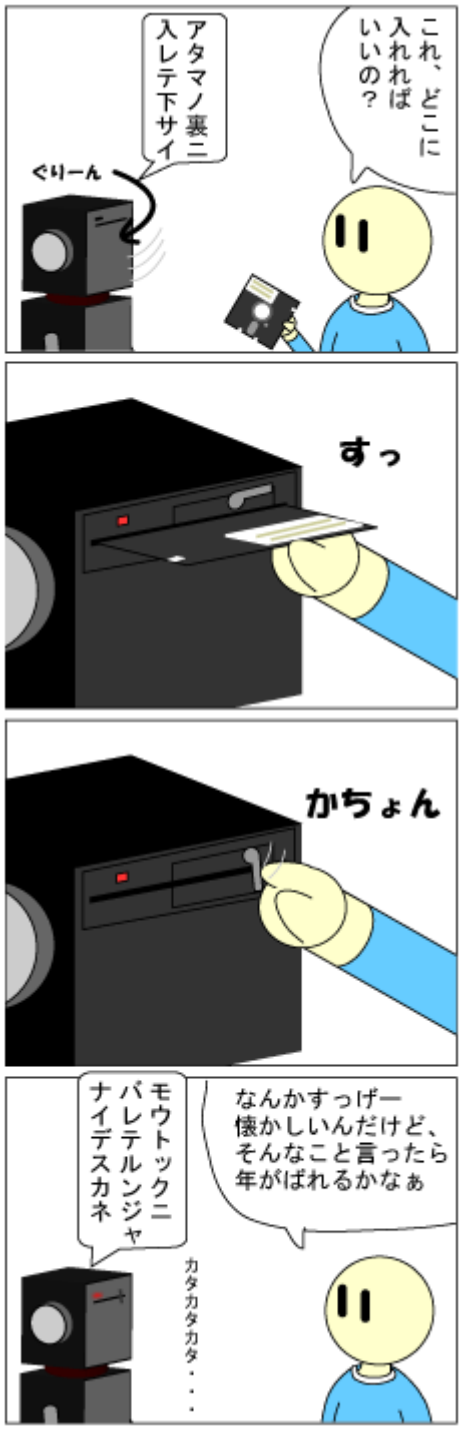
甘党ロボ

「インストール」



さすがに２ＨＤ

「再インストール」

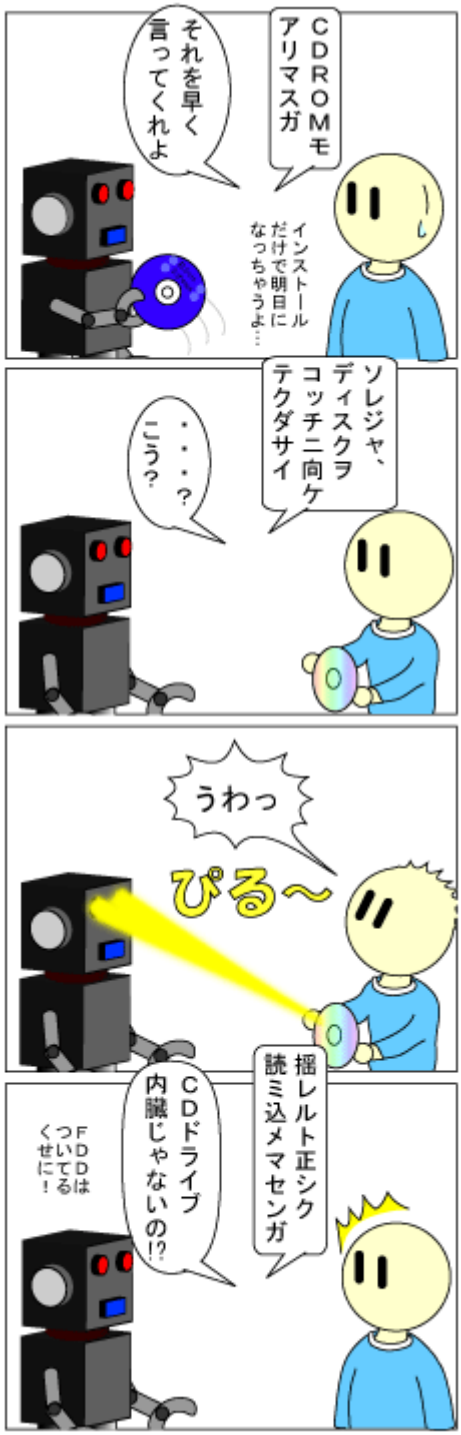


Ａドライブ

「再々インストール」

昔はいつもこんなでしたが←だから年がばれるって

「再々々インストール」



外付け CDROM ドライブ

（転載日付: 2009 年 8 月 6-19 日）

Zed入門 II
- Twitter on Zed -

jscripter

1. はじめに

TSNETスクリプト通信第3号(発行: 2008年11月30日)に「Zed入門」を書いて、約9ヶ月が経過した。本稿では、その間のZedの進展について述べ、Zed上で動作するTwitterクライアントであるPerlスクリプトを紹介する。また、Zed上でスクリプトを目的に応じて適切に動作させるためのスクリプト特別ルールの書き方についても実例に即して述べる。

著者は、デスクトップに存在する情報とWebに存在する情報を統合して、より高度な情報システムを構築することに興味がある。結果として得られる情報をデスクトップに止めるのか、Webに出すのかは内容次第であろう。パーソナルな情報にオープンな情報がぶつかり、相互作用し、溶け合い、反応し、融合することこそが、Webの大海の波が打ち寄せるデスクトップの浜辺における思考を活性化するのである。

2. Zedのその後

Zedは、現在、Ver. 0.41(α)である。前の記事を書いたときは、Ver. 0.35だった。Ver. 0.37でファイル管理ウィンドウが実装されて、日記のテキストなど同一のディレクトリにあるファイル名が類似した同一の拡張子のファイルを編集するのに大変便利になった。

スクリプトを使う場合の動作の不具合の解消やPerl5.8/5.10を使う場合の標準入出力に関わるスクリプトの書き方のノウハウの完成は、作者の久世さんの尽力なくして達成できなかったに違いない。大変感謝している。まず、そのノウハウに関して述べておこう。

Perl5.8/5.10の場合は、スクリプトの出力のバッファのフラッシュ設定を行い、標準入力の終点の判定について次のように書く必要があることがわかった。このようにしないとスクリプトが入力待ちで停止してしまう。jperlの場合にはこのような書き方は不要である。

```
$| = 1;# 出力のバッファのフラッシュ設定

while(<>){
    # 終点判定
    if (index($_, "¥x1A") > -1) {
        last;
    }
    chomp;
    $input = $_;
}
```

3. Twitter

Twitterは2006年頃の登場時に一度話題になったが、使い道がもう一つ見えてこなかった。今年前半にオバマ大統領も使っていると話題なり、最近では、センサーの出力をTwitter経由で取得するような方法が話題になった。Twitterのプログラマブルな読み書きができれば、そのような自動化も可能になる。少し勉強してみようと、Zed上からTwitterを読み書きしてみることにした。

3.1 statusをupdate

参考になるPerlスクリプトに次のようなものを見つけた。興味深いモジュールが満載されたスクリプトである。

> Twitterのコマンドラインクライアント - TECHMemo
> http://d.hatena.ne.jp/dann/20070408/p2

これを参考にして、次のようなスクリプトを書いた。更新日記記事のHTMLタイトル行からカテゴリとタイトル、URLを取得して、Twitterのstatus(小さな140文字からなる記事)をupdateするだけのものである。「このような記事を書きました」とひっそりとささやくのである。

[zed2twitter.pl]

```
#!/Perl5.8/bin/perl.exe
# Title = zed2twitter
# Para =< tmp.log
# Input = tmp.log
# Output =  tmp.log

use strict;
use warnings;

use YAML;
use Encode;
use Net::Twitter;
use File::Spec;
use File::HomeDir;
use Term::Prompt;
use Getopt::Long;
use Term::Encoding;

$| = 1;# for Zed

our $config;
our $twitter;
our $encoding;
our $status;
our $input;
our $baseurl = "http://homepage1.nifty.com/kazuf/renewal.html";
```

```
main();

sub main {
    setup_encoding();
    setup_config();
    setup_client();

    my %methods = (
        update => ¥&update,
    );

    my $method;

    # only on Zed
    while(<>){
        if (index($_, "¥x1A") > -1) {
            last;
        }
        chomp;
        $input = $_;
    }
    ## twitter methods with ddl
    if($input =~ /^(.+?): (.+)$/){
        if($1 eq 'update'){
             $method = 'update';
             $status = decode($encoding, $2);
        }
    ## read a title and a url in an article of renewal.html
    }elsif($input =~ /^<div .+?(¥[.+?¥]).+?<A NAME="(.+?)">(.+?)<¥/A><¥/div>$/){
        $method = 'update';
        $status = decode($encoding, $1 . $3 . " ${baseurl}#" . $2);
    ## read directly text data on Zed
    }elsif($input =~ /^.+$/){
        $method = 'update';
        $status = decode($encoding, $input);
    }else{
        print "No data!!!¥n";
        exit;
    }

    $methods{$method}->();
}

sub setup_encoding {
    $encoding = Term::Encoding::get_encoding();
    $encoding ||= "utf8";
    print "Encoding: ", $encoding, "¥n";
}
```

```
sub setup_config {
    my $path   = File::Spec->catfile(File::HomeDir->my_home, ".twitter");
    $config = eval { YAML::LoadFile($path) } || {};

    my $changed;
    while (!$config->{username} || !$config->{password}) {
        $config->{username} = prompt('x', 'username: ', '', '');
        $config->{password} = prompt('p', 'password: ', '', '');
        $changed++;
    }

    save_config($path, $config) if $changed;
}

sub save_config {
    my ($path, $config ) =@_;
    YAML::DumpFile($path, $config);
    chmod 0600, $path;
}

sub setup_client {
    $twitter = Net::Twitter->new(
        username  => $config->{username},
        password  => $config->{password},
    );
}

# STATUS METHODS
sub update {
    eval { my $response = $twitter->update($status) };
    if( $@ ) {
        warn "update failed because: $@¥n";
    }
    print_update_status();
}

# print
sub print_update_status {
    if($ARGV[1]){
        printf "Set status: %s¥n", $ARGV[1];
    }else{
        printf "Set status: %s¥n", encode($encoding, $status);
    }
}

__END__
```

次のスクリプトの特別ルール行の部分は、Twitterでささやくデータを取得する範囲を読み込み、処理した後に元のデータに書き戻すように動作する。従って、処理する範囲を変化させたくない場合に用いることができる。この特別ルールに基づくスクリプトの標準出力はサブウィンドウに出力される。

```
#!/Perl5.8/bin/perl.exe
# Title = zed2twitter
# Para =< tmp.log
# Input = tmp.log
# Output =  tmp.log
```

次の最初のサブルーチンは、Term::Encodingモジュールを使用しているが、システムのencodingを取得する。Windowsでは、cp932を返す。出力はサブウィンドウに出力される。

```
sub setup_encoding {
    $encoding = Term::Encoding::get_encoding();
    $encoding ||= "utf8";
    print "Encoding: ", $encoding, "¥n";
}
```

次の三つのサブルーチンは、スクリプトを最初に起動すると、Twitterのユーザー名(username)とパスワード(password)をプロンプトを出して問合せ(Term::prompt)、入力データをYAML形式に変換して、ユーザーのホームディレクトリ(Windowsでは、環境変数HOMEの値が入る: File::HomeDir)に、.twitterというファイル(File::Spec)を作成する(YAML::Dumper)。次回のアクセスからはそのデータを読み出して(YAML::LoadFile)、Net::Twitterに使用する。以上、三つのサブルーチンは参考にしたスクリプトにあったものそのままである。大変参考になるスクリプトであった。感謝している。mainのルーチンで、最初にこの三つのサブルーチンを実行している。このスクリプトの構成も元のままである。

```
sub setup_config {
    my $path   = File::Spec->catfile(File::HomeDir->my_home, ".twitter");
    $config = eval { YAML::LoadFile($path) } || {};

    my $changed;
    while (!$config->{username} || !$config->{password}) {
        $config->{username} = prompt('x', 'username: ', '', '');
        $config->{password} = prompt('p', 'password: ', '', '');
        $changed++;
    }

    save_config($path, $config) if $changed;
}

sub save_config {
    my ($path, $config ) =@_;
```

```
    YAML::DumpFile($path, $config);
    chmod 0600, $path;
}

sub setup_client {
    $twitter = Net::Twitter->new(
        username  => $config->{username},
        password  => $config->{password},
    );
}
```

次にmainルーチンの中で、Zed上の編集テキストからデータを取得する部分を抜き出そう。if文の二番目の選択肢は、著者の更新日記(http://homepage1.nifty.com/kazuf/renewal.html)の記事のタイトル部分を範囲指定して、カテゴリ、タイトルとリンクを正規表現によるパターンマッチで取得し、updateを実行する。if文の最初の選択肢は、行頭に「update: status」のように記載した場合には、statusの部分を取得してupdateメソッドを実行する。三番目の選択肢は、特にパターンはないが何か範囲を指定した場合にはその範囲をstatusとして取得してupdateする。いずれの場合も、Zed上のテキストの文字コードはcp932なので、Encodeモジュールにより、cp932でdecodeして、Net::Twitterに渡す。これがポイントである。

```
    # only on Zed
    while(<>){
        if (index($_, "¥x1A") > -1) {
            last;
        }
        chomp;
        $input = $_;
    }
    ## twitter methods with ddl
    if($input =~ /^(.+?): (.+)$/){
        if($1 eq 'update'){
             $method = 'update';
             $status = decode($encoding, $2);
        }
    ## read a title and a url in an article of renewal.html
    }elsif($input =~ /^<div .+?(¥[.+?¥]).+?<A NAME="(.+?)">(.+?)<¥/A><¥/div>$/){
        $method = 'update';
        $status = decode($encoding, $1 . $3 . " ${baseurl}#" . $2);
    ## read directly text data on Zed
    }elsif($input =~ /^.+$/){
        $method = 'update';
        $status = decode($encoding, $input);
    }else{
        print "No data!!!¥n";
        exit;
    }
```

3.2 user_timelineからstatusをリストアップ

Net::Twitterにはいろいろなメソッドが実装されているが、自分が欲しい形でデータを取得したいというような場合には、モジュールの使い方に精通する必要がある。精通したとしても融通が利くかどうかは定かではない。それよりも直接的に問題を解決しようというのが次のスクリプトである。Twitterのuser_timelineを取得して表示する。

[twemo.pl]

```
#!/Perl5.8/bin/perl.exe
# Title = twemo
# Para = -h
# Output

use LWP::Simple;
use HTML::Entities;
use Encode;
use Date::Parse;
use Date::Format;

my $opt = shift(@ARGV);       # get options
my $screen_name = 'jscripter';# option: setup a screen_name
my $opth = 0;                 # option: add hyperlink tags
my $count = 20;               # option: a number of viewing statuses

if($opt =~ /-s([^-]+)/i){
    $screen_name = $1;
}
if($opt =~ /-h/i){
    $opth = 1;
}
if($opt =~ /-c(¥d{1,3})/i){
    $count = $1;
}

my $user_statuses = get("http://twitter.com/statuses/user_timeline.xml" .
                        "?screen_name=$screen_name&count=$count");

my @statuses = (
    $user_statuses =~ /<status>.*?
                       <created_at>(.*?)<¥/created_at>.*?
                       <text>(.*?)<¥/text>.+?<¥/status>/sgix
);

foreach (@statuses){
```

```
    if(my $time = str2time($_)){
        my @lt = localtime($time);
        my $timestr = strftime("%c %z", @lt);
        print  "${timestr}: ";
    }else{
        # -h option
        if($opth){
            $_ =~ s/(http:¥/¥/[^¥s]+)¥s*$/<a href="$1">$1<¥/a>/g;
        }
       $_ = encode('cp932', decode_entities($_));
        print "$_¥n";
    }
}
```

このスクリプトの特別ルールは次のようである。Paraはコマンドラインオプションの指定である。-hはURLにハイパーリンクのHTMLタグを加えて出力するオプションである。Inputはないのでzedからの入力はない。単にカレット位置にスクリプトの出力をOutputするというものである。すなわち、TwitterのUser_timelineから取得したデータをZedの編集ウィンドウ上のカレット位置に整形して挿入出力する。

コマンドラインでもオプションをつけて動作可能である。

```
#!/Perl5.8/bin/perl.exe
# Title = twemo
# Para = -h
# Output
```

LWP::Simpleモジュールのgetメソッドで、user_timelineを取得する部分は次の通り。ユーザー名(screen_name)と取得する記事(status)の数(count)をCGIの形式で設定する。このTwitterのAPIは非常に詳しく公開されている。次のURLを参照すること。

Twitter API Wiki / Twitter API Documentation
http://apiwiki.twitter.com/Twitter-API-Documentation

Twitter API Wiki / Twitter REST API Method: statuses user_timeline
http://apiwiki.twitter.com/Twitter-REST-API-Method%3A-statuses-user_timeline

```
my $user_statuses = get("http://twitter.com/statuses/user_timeline.xml" .
                        "?screen_name=$screen_name&count=$count");
```

さて、このgetメソッドで取得したXML(user_timeline.xml)のstatus(記事の項目)の一つの単位を示しておこう。

```
<status>
  <created_at>Wed Aug 19 14:30:27 +0000 2009</created_at>
  <id>3405832828</id>
  <text>[&#21315;&#22812;&#21315;&#20874;]&#26360;&#29289;&#12398;&#21487;&#33021;&#24615;
http://bit.ly/lKf4Q</text>
  <source>&lt;a href="http://search.cpan.org/dist/Net-Twitter/" rel="nofollow"&gt;Perl
Net::Twitter&lt;/a&gt;</source>

  <truncated>false</truncated>
  <in_reply_to_status_id></in_reply_to_status_id>
  <in_reply_to_user_id></in_reply_to_user_id>
  <favorited>false</favorited>
  <in_reply_to_screen_name></in_reply_to_screen_name>
  <user>
    <id>6162172</id>

    <name>jazzy scripter</name>
    <screen_name>jscripter</screen_name>
    <location></location>
    <description></description>
    <profile_image_url>http://a1.twimg.com/profile_images/360266004/hiroshima_night_2009-
01_crop_normal.jpg</profile_image_url>
    <url></url>
    <protected>false</protected>

    <followers_count>6</followers_count>
    <profile_background_color>709397</profile_background_color>
    <profile_text_color>333333</profile_text_color>
    <profile_link_color>FF3300</profile_link_color>
    <profile_sidebar_fill_color>A0C5C7</profile_sidebar_fill_color>
    <profile_sidebar_border_color>86A4A6</profile_sidebar_border_color>

    <friends_count>4</friends_count>
    <created_at>Sat May 19 15:37:37 +0000 2007</created_at>
    <favourites_count>0</favourites_count>
    <utc_offset>32400</utc_offset>
    <time_zone>Tokyo</time_zone>
    <profile_background_image_url>http://s.twimg.com/a/1250203207/images/themes/theme6/bg.gif</p
rofile_background_image_url>

    <profile_background_tile>false</profile_background_tile>
    <statuses_count>40</statuses_count>
    <notifications></notifications>
    <verified>false</verified>
    <following></following>
  </user>
</status>
```

肝心なところは次の部分である。created_atタグに投稿時間が、textタグに記事(status)が記載さ

れている。status毎に、この二つのデータを取得すれば良い。もう一つ注意しておく必要があるのは、textタグのUnicode文字のテキストデータは実体参照(Entity)にエンコードされていることだ。ブラウザでは読めても人間には読めない。実体参照をデコードする必要がある。

```
<status>
  <created_at>Wed Aug 19 14:30:27 +0000 2009</created_at>
  <id>3405832828</id>
  <text>[&#21315;&#22812;&#21315;&#20874;]&#26360;&#29289;&#12398;&#21487;&#33021;&#24615;
http://bit.ly/IKf4Q</text>
.....

</status>
```

---

statusタグの中のcreated_atタグとtextタグの値を取得する正規表現によるパターンマッチを考えてみる。getメソッドではXMLファイル全体を取得しているので、改行なども含まれたテキストファイルである。これから一気に取得する。そのためには、sオプションを使う。改行も「.」にマッチするので文字列全体を1行と見なしてマッチさせることができる。

gオプションでマッチがある限り繰り返してマッチする。これを配列コンテキストで評価すると配列にマッチした順に格納されていく。後は適切に取り出して加工するだけである。textタグの値は、HTML::Entitiesモジュールを使ってデコードする。

下記の正規表現は複数行に渡って記述されているが、xオプションで可能となっている。正規表現のすべては空白を除いてつながったパターンとして評価される。

このマッチの評価の問題点は、created_atタグとtextタグの順序が前後するとマッチしなくなることである。マッチしなくなれば、出力が変化したことを疑わなくてはならない。無論、二つのタグのマッチの問題であるから、二種類のパターンを準備して対応しておくことも可能である。

```
my @statuses = (
    $user_statuses =~ /<status>.*?
                       <created_at>(.*?)<¥/created_at>.*?
                       <text>(.*?)<¥/text>.+?<¥/status>/sgix
);
```

グローバル・パターンマッチからデータを取得するために、whileループを使ったスクリプトを次に示す。

[twemo2.pl]

```
#!/Perl5.8/bin/perl.exe
# Title = twemo2
```

```
# Para = -h-r
# Output

use LWP::Simple;
use HTML::Entities;
use Encode;
use Date::Parse;
use Date::Format;

my $opt = shift(@ARGV);       # get options

my $screen_name = 'jscripter';# -s[screen_name] option: setup a screen_name
my $opth = 0;                 # -h option: add hyperlink tags
my $count = 20;               # -c[number] option: a number of viewing statuses
my $rev = 0;                  # -r option: reverse statuses with time-order

if($opt =~ /-s([^-]+)/i){
    $screen_name = $1;# set a screen name: screen_name after -s
}
if($opt =~ /-h/i){
    $opth = 1;# output hyperlinks of urls: 1
}
if($opt =~ /-c(¥d{1,3})/i){
    $count = $1;# set a count: single digit to triple digits after -c
}
if($opt =~ /-r/i){
    $rev = 1;# reverse mode: 1
}

my $user_statuses = get("http://twitter.com/statuses/user_timeline.xml" .
                        "?screen_name=$screen_name&count=$count");

my %statuses = ();
while(
    $user_statuses =~ /<status>.*?
                       <created_at>(.*?)<¥/created_at>.*?
                       <text>(.*?)<¥/text>.+?
                       <¥/status>/sgix
){
    my @lt = localtime(str2time($1));
    my $timestr = strftime("%c %z", @lt);
    $statuses{$timestr} = $2;
}

if($rev){
    @sorted_keys = reverse sort keys %statuses;
}else{
    @sorted_keys = sort keys %statuses;
}
```

```
foreach my $created_at (@sorted_keys){
    print  "${created_at}: ";
    # -h option
    if($opth){
         $statuses{$created_at} =~
             s/(http:¥/¥/[^¥s]+)¥s*$/<a href="$1">$1<¥/a>/g;
    }
    my $status = encode('cp932', decode_entities($statuses{$created_at}));
    print "$status¥n";
}

__END__
```

パターンマッチでデータを取得するのは次の部分である。whileの条件文のなかで、グローバルなパターンマッチを行うと、パターンマッチする度にループが回るので、連想配列に投稿時間(created_at)の値をキーにして格納していく。この場合、時間をローカル時間に変換してキーとしている。Date::ParseとDate::Formatモジュールを使っている。

スクリプトで-rオプションを指定すると、連想配列のキーをソート(sort)して逆順に並べ替えた配列を作る(reverse)。その配列の値をキーにして連想配列の値を取り出せば、逆順にソートした値を取り出すことができる。

```
my %statuses = ();
while(
    $user_statuses =~ /<status>.*?
                       <created_at>(.*?)<¥/created_at>.*?
                       <text>(.*?)<¥/text>.+?
                       <¥/status>/sgix
){
    my @lt = localtime(str2time($1));
    my $timestr = strftime("%c %z", @lt);
    $statuses{$timestr} = $2;
}

if($rev){
    @sorted_keys = reverse sort keys %statuses;
}else{
    @sorted_keys = sort keys %statuses;
}
```

スクリプトのオプションの設定方法や出力を整形して取り出す部分などは、スクリプトを読めば明らかなので特に説明しない。

コマンドラインではオプションは実行するたびに変更が可能だが、Zedのスクリプトの場合は、よく使うオプションを設定したスクリプトをTitleを変えて複数置いておくのが良い。編集からTitleで実行するスクリプトを選ぶことができる。スクリプトは、Zedのインストールディレクトリの¥macro¥exec¥menu¥editに置くことになっている。詳細は、ヘルプのScript仕様.txtを参照すること。

4. 最後に

以上のように、Zed上でスクリプトを動かして、インターネット上でささやいたり、ささやきを読み込んだりすることは自在である。本稿が読者のインターネットライフに少しでもご参考になれば幸いである。

最後に、Zedを提供していただき、開発を続けられている久世さんに感謝してキーボードから手を離すことにしたい。いつも要望をご検討いただきありがとうございます。引き続きがんばってください。新しいバージョンを楽しみにしています。

（投稿日付：2009年8月22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。みんなの記事は既にセットアップ済みだ。

今回は、編集にScribusというOpen source desktop publishing ソフトウェアを使ってみようと考えていたのたが、しばらく触って断念。簡単ではない。OpenOffice.orgのWriterに引き続きお世話になることにした。

今号こそは、ニュースを載せようと考えていたのだが、Twitterネタで代わりにしよう。MVCフレームワークネタも再度書くには具体性が必要と考えて、お蔵入り。PythonもMVCフレームワークネタに合わせて何か書くつもりだったが、具体化までには至らなかった。実際に役に立つレベルに持っていくためにはニーズもあるし、Perlで済むのだったら、そちらで間に合わせるのが楽だしと^^;)いうことになる。

スクリプティングの世界もどのように変化するのか、Rakudo Perl 6の動きに期待している。しかし、うまく行ったとして、実際にフルに使うようになるにはさらに5年ぐらい掛かるのではないかという気がしている。ソフトウェアもモノと同様に熟成するには10-20年ぐらい掛かるものらしい。

(投稿日付: 2009年8月22日)

ＴＳＮＥＴスクリプト通信

2009年8月22日　2.2.001版発行
2009年8月22日　2.2.000版発行

投稿規程

TSNETWiki ： 「投稿規程」のページを参照のこと

編集委員会(投稿順)

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

使用許諾・配布条件

免責事項

編集ソフトウェア

OpenOffice.org 3.1.0 Writer

発行所(一次配布所)

TSNETWiki ： 「TSNETスクリプト通信」のページを参照のこと
