MacPerl 入門

まかせて!!

MacPerl

編集者・デザイナー・DTPオペレーターのためのMacPerl（1）

CD-ROM

市川せうぞー●有限会社シンクス

テキスト，XPressタグ，Illustrator，PostScript，PDF，HTML，SGML，XML……。DTPの周辺のあらゆるカベを取り払い，これらすべてを結び付けるPerl。その魅力とパワーを紹介する。第1回目はインストールとOver View。そう，MacPerlをインストールしなくちゃはじまらない。“Metal Glue is it True.”

Howdy!

さて，さっそく読者をふるいにかけさせていただこう。

Perlは便利なんである。この便利さが何を意味するのか骨身に染みてわかっておられる御仁は，この特集を読む必要はまったくない。あるいは，便利なことがお嫌いな哲学者もお断りだ。また，宗教上の戒律から禅問答に終始したい御房も退室いただこう。

便利なことはただそれだけで価値のあることなのだ。有史以来，便利なことは不便なことを常に淘汰してきたといえる。電球が発明されてから，蛍の光で勉学に励むことに価値を見いだせなくなったのと同様だ。そもそも，データ以外に最も価値のある資産は人間の労力であるからして，いかに労を要せずデータを作成し，利用するかがネアンデルタールの昔から我々の一番の関心事であることは言うまでもない。

人間はちょっとでもラクをしたい「無精」な生き物だと言える。しかし恥じることは耳かきのさじ程もない。むしろPerlでは「無精」「短気」「傲慢」は3大美徳とされているのだから。9の力で10のことを成し遂げれば，このバランスの上においてあなたの成果は正当に評価されるだろう。しかしPerlを使えば，1の力で（言いかたを変えるなら，たった1行で）何万もの成果を叩き出すことができる。

ではおきまりのPerl流の挨拶を！

```
print "Howdy, Perl World!\n";
```

何をするか，何をすべきか？

お客様は神様なんである。このキャッチはもう何十年も前にとある演歌歌手が流行らせた「奴隷契約」であるが，どっこいこの社会にはたち

What is Perl?

●Perlってなんだ？

Perlは1987年，Larry Wall氏によって生み出されたテキスト処理言語で，正式にはThe Practical Extraction Report Language（実用的情報抽出レポート言語）の略である。実際にはその多重人格的性格ゆえに，さまざまな分野・用途で使われている。最近ではWEBのCGI処理を中心としてたいへん人気の高い言語となっているのを知っている読者もいるかもしれない。

驚くべきことに，Perlと名の付くプロダクツはすべてオープンソースでパブリックドメインになっており，世界中のだれもが，使いたい時にダウンロードして自由に使用することができる。

バージョン・アップの度に請求書を回してくるどこかの新興宗教とはえらい違いだ。オープンソース文化の根底には便利なモノを共有しようという精神にあふれているんである。

●MacJPerlって？

PerlはもともとUNIX環境で生まれたが，さまざまな方の努力によって，いまではほとんどのプラットフォームで動作するPerlが存在する。Macintoshも例外ではなく，「MacPerl」として移植・拡張されている。書き方さえちょっと気をつければ，WindowsでもUNIXでもMacintoshでも動作するプログラムを書くことができる。

Perlは標準で1バイトの文字の処理を想定してつくられている。ところが日本語の仮名や漢字はご存じのように2バイトで1文字を表現しているため，2バイト目に「\」などPerlにとって特別な意味を持つ文字が来てしまった場合，誤作動や誤処理の原因になる。また，2バイトは2文字とカウントされて1文字として認識されないという問題もある。これらの問題に対してさまざまなアプローチがあるが，アプリケーション的にこれらの問題を意識することなく，標準で2バイト文字を扱える「MacJPerl」というタイプが存在し，この連載でもMacJPerlを用いて説明を進めている。本文中では「Perl」や「MacPerl」と表記してもこのMacJPerlを指していることがほとんどである。

現人神と遭遇す

中学生のころ，デパートの屋上に太田宏美さんのサイン会に行ったことがある。その夜は感激のあまり寝付くことができなかった……。そんな遠い記憶の彼方に忘れ去られていた感覚が，突然呼び覚まされたのだった。

11月11日，オライリー・ジャパンの主催で行なわれた「Perlカンファレンス」に，Perlの生みの親であるLarry Wall氏が特別講師として来日されたのである。雲の上の先生方のさらに上空，成層圏に住む「現人神」との遭遇に，あたしの感激がいかほどのものであったか？熱烈なMacファンの方々には申しわけないが，Steve Jobsに逢ったってこんなに感激しないだろう。

この時あたしは，ミーハーそのもの・ミーハー権化であったと思う。

あたし●「Can I Take a Picture?」
現人神●「Sure!」

なんというぶっきらぼうで，野蛮なミーハーであろうか？　しかし現人神はさらに「together」とおっしゃったのだったよ。全身の血が逆流したね。

あたしは，足は震えてくるし，顔はこわばるしで世間様にお見せできるような写真ではないが，ともかくLarry先生のこの笑顔に人格すべてがにじみ出ているような気がするのは気のせいか？

こういうカメラ持参の物見遊山な出席者はあたしぐらいなものだったのだろう。おかげでカンファレンスそのものはレベルが高すぎてさっぱりついていけなかった。まあ，前田薫先生と歌代和正先生の御尊顔を拝しただけでもよしとしよう。あぁ，やっぱりミーハーだ……。

どころにマッチングして，商取引の対等で正当なデモクラシーを排除してしまった。

この業界だって例外じゃない。クライアントが多用する半角カナがたとえWin機のデフォルトだとしても，それに文句を付けることは信教の自由に反するのだ。クライアントが英数字の半角全角が判別できないワープロを愛用していたとしても（あぁ，不幸なことにこのワープロには罫線できれいに表組みできるすばらしい機能もついている！），笑顔で受け取るのが我々の仕事なのだ。もちろん，これらを清書する対価は請求書には含まれていないのであるが。

たとえば，このページの本文の組版をすべてPerlで処理したといったら驚くだろうか？　この本文にはいくつかの要素が含まれている。見出しや太字，イタリックやコード文，上付インデックスや脚注……。「カッコ」や・（中黒），リーダやダーシなどのツメを調整するカーニング。（QuarkXPress 3.3Jでは実現できない）拗促音や音引きの行頭禁則にいたるまで制御できる（もちろん訂正だってOKさ）としたらどうだろうか？

これまで，編集者やデザイナーは色彩学の教授よろしく48色のマーカーを使用して，微妙な色を使い分けて入念な指示をしていたと思う。間違えたら赤ペンで直したりして……。現場のオペレーターはこの巧妙ないやがらせに豚のように文句を垂れながら迎えた朝が……いや，思い出すのはもうやめよう。

Perlを使えばこれらの処理にかかる時間は数秒になる。さらに，この組版に使用したプログラムをホンのちょっと書き換えれば，HTMLを書き出すなんてことも朝飯まえなのだよ。ウソだと思うならどちらもCD-ROM MACLIFEに収録されているので確かめてみるといい。

この連載のスタンス

最初にちょっとだけいいわけをさせてもらうことにする。この連載でPerlのすべてを紹介することはとうていできない。紙ひこうきで月を目指すのと等しい。Perlの歴史や背景を説明するだけでも数ページでは足りないだろう。第一，あたし（※1）は一介の（DTP）Perlユーザーなんであり，ライターを生業とするわけでも，Perlを理論立てて説明できるような立場でもないことをご理解願いたい。

※1　この連載では筆者を表わす一人称を「あたし」と表記します。これも「傲慢」ゆえになせる業とお許しください。

行儀作法にうるさい先生がたのおしかりを恐れずに言わせていただければ，この連載はPerlの魅力をおもしろおかしく実用的に紹介するという点につきる。どんなアプリケーションも使っただけの便利さしか得られないし，使っただけの便利さが得られれば道具としては必要十分なんである。もし，さらにあなたの興味を引くことに成功したのなら，後に挙げる参考書籍などでPerlに本格的に取り組んでいただきたい。あなたの努力と期待を裏切ることなく，あなたのPerlは成長するのだから。

この連載を通じてホンの少しでもPerlの間口が広がれば，あわよくば出先のマシン環境にもインストールされて，わざわざ（断って）インストール作業をしなくても済むようになれば（笑），あたしの無精心も満たされるし，これまでお世話になりっぱなしのPerlコミュニティの方々にもせめてもの恩返しができるというものだ。

さあ，Perlの扉をこじあけよう！

インストール

MacJPerlは漢字Talk 7以上のMacintoshで動作する。最新のMac OS 8.5でも大きな問題は報告されていない。FATバイナリのため，68KでもPowerPC上でも同様に動作する。

あなたにとって必要かつ十分な材料はすでに用意してある。以下の手順通りにインストールしていただきたい。

❶CD-ROMに添付されている「Mac-_Perl_520r4_appl」と「MacJPerl 5.1.5r4.sea」をHDにコピーする。
❷Mac_Perl_520r4_applはインストーラになっているので，ダブルクリックして起動する。
❸任意の場所を選んでインストールをする。
❹MacJPerl 5.1.5r4.seaをダブルクリックするとMacJPerlアプリケーションが解凍される。
❺いま解凍したMacJPerlアプリケーションを手順❸でインストールした英語版MacPerlアプリケーションと入れ替える（※2）。

※2　英語版MacPerlは捨てないで，圧縮してどこかに取っておくといいかもしれません。というのは，MacJPerlでは動作しないパッケージやライブラリがあるからです。ただし，アプリケーションのままどこかに置いておくと，プログラムを起動させたとき，MacJPerlとMacPerlのどちらが起動するか確定されません。どちらかを起動させておき，プログラムを実行すればいいのですが，AppleScript経由で処理させるときなどに不都合があります。

❻CommandキーとOptionキーを押しながら再起動してデスクトップの再構築をする。

MacJPerlアプリケーションをダブルクリックして起動してみよう。実際にPerlを動作させる前に，やらなければならないことがある。

MacPerl 入門

まかせて!! MacPerl

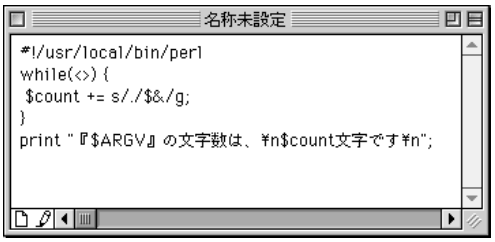


図1●打つべし、打つべし！

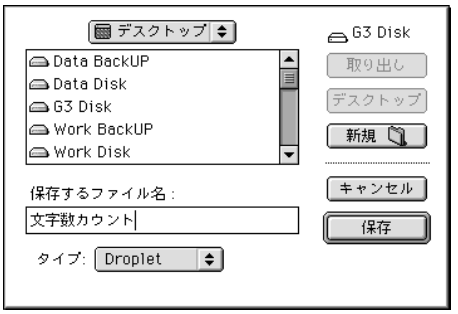


図2●DropletはMacintoshならではの使いやすさだ

文字数カウント

図3●これで立派なアプリケーションだ

❶編集メニューから「編集様式」を選びフォントを“Osaka―等幅”などに設定する。
❷編集メニューから「環境設定」を選び，ライブラリフォルダ（フォルダ名は“lib”，アプリケーションと同じ階層に入っている）が正しく選ばれていることを確認する。

これで準備が整った。

文字数をカウントする

さっそく，カンタンで実用的なプログラムのを紹介する。そう，習うより慣れろ。考えるよりタイプしろ。プログラム修得に近道はない。

手始めにテキストファイルの総文字数を計算させてみることにしよう。

MacJPerlのファイルメニューから「新規」を選ぶか，Command＋Nを押すと新規ウィンドウ（Perlエディタ）が開く。ここに下記のように打つべし，打つべし（図1）。

```
#!/usr/local/bin/perl
while(<>) {
 $count += s/./$&/g;
}
print "『$ARGV』の文字数は，➡
    \n$count文字です\n";
```

注）スクリプト内の➡印は，改行せずに下の行が続くというマークです。

Perlエディタに上記のとおりタイプしおえたら，

❶ファイルメニューから「別名で保存」を選ぶか，Command＋Shift＋Sを押すと図2のような保存ダイアログが出る。
❷名前を入力したらタイプを保存形式を「Droplet」（ドロップレット）にして（※3）任意の場所に保存する。

※3　MacPerlでは「実行専用」という，面白い保存形式があります。この形式で保存すると，MacPerlアプリケーション本体のリソースをコピーして持つため，MacPerlアプリケーション本体がインストールされていない環境でもランタイムとしてふるまいます。ただし，ファイルサイズが大きくなり，配布用としては一長一短です。もちろん，Macintosh環境以外ではこういう機能はありません。もし，他の（Windowsなどの）環境で動作させたいときはテキストファイルとしてスクリプトのみをもっていけばよいでしょう。通常のMac上であれば，このドロプレット形式が最も便利です。

❸図3のようなファイルができたことを確認する。

MacPerlではスクリプトはドロップレット（※4）になっているのが普通だから，コマンドを打ち込んで処理ファイルを指定する必要はない。処理させるファイル（この場合は総文字数を調べたいファイル）をいま作ったドロップレットにドラッグ＆ドロップするだけなんである。実に便利で理にかなった方法ではないか。あっという間に，文字数が表示されるんである。

※4　標準でインストールされていると思いますが，作ったプログラムにドラッグ＆ドロップ（ドロップレット）するためには，AppleScriptがインストールされていなければなりません。

ちなみに，このドロップレットを再編集するためには，MacJPerlのアプリケーションにドラッグ＆ドロップするか，Optionキーを押しながら，ダブルクリックすればOKだ。

いまここではスクリプトの詳しい解説はしない。とりあえずここでは，初めて書いたプログラムが実際に動いていることに素直に感動していただきたい。なにせ，あたしはここまで来るのに何カ月もかかってるんだ（笑）。今回はOver Viewということで，ひととおりの紹介のみにしたい。パンチドランカーになってしまったら困るしね。

検索文字を含む行を表示する

ある文字列が含まれたファイルを探すのは，結構手間だったりする。Mac OS 8.5からはインテリジェントな「シャーロック」なる探偵を雇ったが，前調査が長すぎて目的のファイルへたどり着くまでの道のりはなかなか険しいようだ。毎日クライアントから山のように持ち込まれるMOにいちいちそんな手間は掛けられない。第一，目的のファイルはおよそどこにあるかぐらいは見当がついている。そんな時にこのスクリプトは便利だ。検索したい複数のテキストファイルをドラッグ＆ドロップすると，入力した検索文字列に該当する行をすべて表示する。

```
#!/usr/local/bin/perl
print "検索文字を入力してください\n";
$find_words = <STDIN>;
chop($find_words);
while(<>){
 if (/$find_words/) {
    print "$ARGV $_";
 }
}
```

じつはこの検索プログラムは隠された力を持っている。普通の文字列検索以外にさらに便利な「正規表現」が使用できることである。もちろん前述の探偵にはそのような芸当はできない。

無駄話が多いせいか，そろそろ紙面の残りが少なくなってきた。反省――。正規表現については次回くわしくお話することにしよう。正規表現を使った検索置換が使えるようになれば，あなたはPerlを手放せなくなるはずだ。

予告

この連載の今後の予定をまとめてみる。

❶MacJPerlの導入とOverView
❷正規表現とパターンマッチ
❸制御構造
❹配列・関数
❺MacPerl拡張とAppleScript
❻ケーススタディ（XPressタグ，Illustrator，Postscript，HTML）

どれもこれも濃いので，予定どおり進むかどうか相当不安なんであるが，乗りかけた船だと思って最後までお付き合い願えれば幸いである。毎回CD-ROM MACLIFEに実用的なスクリプトを収録するので，それを使うだけでもいいし，カスタマイズできればなおいい。冗長なプログラムへの抗議のメールも歓迎しよう。すべてはPerlの扉を開けたあなたのために…。

それでは，次回をお楽しみに。

参考文献 Perlに関する情報源

●書籍

『プログラミングPerl 改訂版』
Larry Wall他，近藤嘉雪訳（オライリー・ジャパン）

Perlを学ぶ上で最もアカデミックな原著。リファレンスとしても秀逸で必携の1冊。電車内などでは重くて腕が痺れてしまうのが唯一の欠点。

『Perl入門』
Ellie Quigley，武舎広幸 他訳（プレンティスホール出版）

スクリプト作成の基礎からプロセス間通信までが平易に綴られており，初心者向き。本全体はUNIX環境を前提としているが，訳者の方がMacPerlユーザーであることと，導入部にほんの少しMacPerlの簡単な導入方法も書かれており心強い。

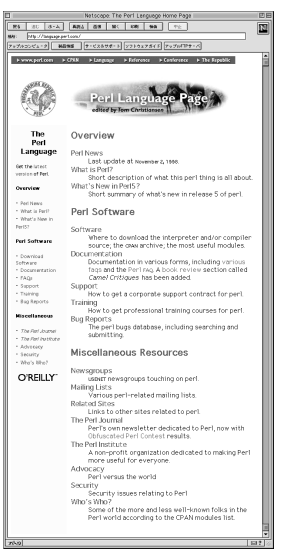
『日本語TEXT加工入門ハンドブック』
『日本語TEXT加工実践ガイドブック』
『日本語TEXT加工実用リファレンス』
中島 靖（情報管理）

ふんだんなスクリプト例とリファレンスは丁寧で秀逸。表紙の昆虫の絵の意味は不明。

●Webサイト

The Perl Language Home Page
http://www.perl.com/perl/

Perl公式サイト。さまざまなソース，スクリプト，FAQ，ドキュメントが参照できるCPAN（Comprehensive Perl Archive Network；総合Perlアーカイブネットワーク）へもここから行くことができる。迷子にならないよう注意が必要。



MacJPerl 五
http://world.std.com/~habilis/macjperl/MacJP5.j.htm

MacJPerl最新版がダウンロードできる。ツール類のリンクもここから取りにいける。

Hora's Home Page [Powered By Mac-OS]
http://www2a.biglobe.ne.jp/~hora/

山本武さんが運営されているページ。MacPerlやAppleScriptを使ったソリューションを提供。会議室を通じたサポートも活発。

MacでPerlのページ
http://www.bekkoame.or.jp/~syunji/perl/mac/macperl.html

Perlに関してのさまざまな話題，Tipsがちりばめられている。

AppleScript Irregulars
http://www.dd.iij4u.or.jp/~xyz/

鎌田幸雄さんの運営されているページ。AppleScriptとDTPの話題が中心だが，MacPerlとQuarkXPressを連携させた情報は見逃せない。

Show Time
http://www.asahi-net.or.jp/~ym3s-ickw/showtime.html

わたしのページ。この連載で紹介したスクリプトを順次掲載します。あっ，更新しなくちゃ。

●BBS，News

NIFTYSERVE

・DTPフォーラム（FDTP）15番会議室せどおうくぱある(SED,AWK,PERL)
・FGAL・テキスト＆スクリプト7番会議室基本処理｜Perl天国1/便利でらくちん!!
・FGAL・テキスト＆スクリプト9番会議室OS依存｜Perl天国2/Win32,Mac等の使い方

NewsGroup（日本語）

・japan.comp.lang.perl
・fj.lang.perl

NewsGroup（英語）

・comp.lang.perl.misc
・comp.lang.perl.announce
・comp.lang.perl.modules

MacPerl 入門

まかせて!! MacPerl

編集者・デザイナー・DTPオペレーターのためのMacPerl (2)

CD-ROM

市川せうぞー●有限会社シンクス

テキスト，XPressタグ，Illustrator，PostScript，PDF，HTML，SGML，XML……。
DTPの周辺のあらゆるカベを取り払い，これらすべてを結び付けるPerl。
その魅力とパワーを紹介する。
第2回目は正規表現とパターンマッチ。
今月号からでも遅くない！
"Get it ON!!!"

前回はMacJPerlの導入と，簡単なスクリプトを挙げてほんのさわりを紹介させていただいた。前回の最後に予告したとおり，今回は「正規表現」を含めたパターンマッチの紹介することにしよう。

正規表現とは

「正規表現」という言葉を初めて耳にする読者もいるかもしれない。正規表現を一言で説明するのはちょっと難しいのだが，いくつか例題を見ていただければすぐに理解できるだろう　。日常的に馴染みのある「YooEdit」や「Jedit」などにも実装されていて（※1），使い慣れればこれほど便利な検索方法はない。「Perlがすばらしいのではなく正規表現がすばらしいのだ」と大袈裟なことを言う人さえいるほどだ。

※1　正規表現にも方言が若干存在していて，表の表現がそのまま使えるわけではありません。それぞれのアプリケーションのマニュアルを参照してください。

次ページに正規表現の一覧をあげておく。よく使うものをいくつか説明しよう。

「.」は改行以外の1文字とマッチする。「愛.県」と書けば，愛知県にも愛媛県にもマッチする。しかし「.」はどんな文字にもマッチしてしまい，「慈愛と県政」などというウソ臭くとんちんかんな文字列にもマッチしてしまうことになりかねない（笑）。これを避けたい場合は，「愛[知媛]県」と書くか「愛(知¦媛)県」と書けばよいわけなんである。「[ ]」の使い方としては他にも，「歩[かきくけこい]」で動詞の活用などを表現したり，「[^～～]」のようにその文字集合以外を表現することも可能だ。たとえば，[^亜-龠]で漢字以外という書き方もできたりする。

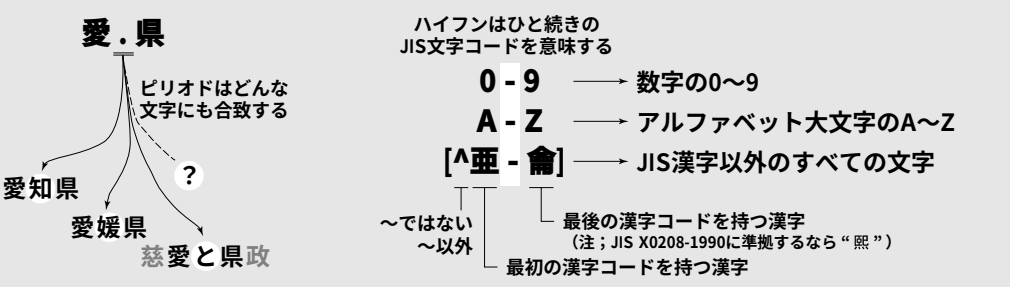
同じ種類の文字が何度も繰り返されるようなパターンも表現できたりする。「?」は直前の文字が0回か1回ある場合にマッチし，「Pea?rl」と書けばPearlとPerlを同時に検索に掛けられる。「*」は直前の文字が0回以上，「+」は直前の文字が1回以上を表す。「Coo+l」と書けば，CoolにもCooolにもCooooooolにもマッチする（クールな説明とはいいがたいが…）。特に制約なく「.+」と書けば，通常は行頭から行末を意味する。これを「最左最長一致」といい，1行中の最も左から可能な限り長くマッチする仕組みだ。Perl5からは「*?」と「+?」の組み合わせが最短の一致として取り入れられた。たとえば，1行中にカッコが2組以上出てくる場合，『(なんとかかんとか）～～（笑)』を（.+）と検索しようとすると『(なんとかかんとか）～～（笑)』すべてがマッチしてしまうが，(.+?）とすれば『(なんとかかんとか)』までしかマッチしない。もちろん，Perl4にはない機能なので，([^)]+）という書き方で回避してきた。Perl4.xで稼動しているマシンも多いため，できるだけ互換の取れる書き方が好ましい。

特定の文字の個数でマッチさせることも可能となっている。たとえば東京都区内の電話番号は「03¥-¥d{4}¥-¥d{4}」で表現できる。

「[ ]」の外にある「^」は行頭を，「$」は行末を表し，これが意外に重宝したりする。「^「」と書けば行頭の"「"だけにマッチし，「。$」と書けば行末の"。"にしかマッチしない。「^$」と書けば空行を表し，「^¥s*$」とすれば，スペースだけの行にマッチする。

意外とカンタンで便利なことが分かっていただけたと思う。これらのパターンを駆使すれば，複雑な文字列パターンも1度で検索対象にできるわけなんである。アルファベットは26文字×大文字・小文字で52文字，さらに数字を足せば62文字あるが，これらを一度ずつ検索するとすればたいへんな労力だが，「[A-Za-z0-9]」なら1回で済んでしまうんである。

正規表現に使われるこれらのメタキャラクタそのものを表現することはできるのかと心配する必要はない。Perlには簡単にメタキャラクタを表す方法がある。単にメタキャラクタの前に「¥」を置くだけだ。半角のパーレンを表したいなら「¥(」と書けばよく，ハイフンを表したいのなら「¥-」と書けばよい。



正規表現の主なメタキャラクター一覧表

●ある特定の文字を表すもの	
.	改行キャラクタ（¥n）以外の任意の1文字
[A-Z0-9]	[ ]の中の任意文字の集合（この場合，アルファベットの大文字か数字の1文字を表す）
[^A-Z0-9]	[ ]の中の文字以外の集合（この場合，アルファベットの大文字か数字以外の1文字を表す）
¥d	数字（[0-9]と同じ）
¥D	数字以外（[^¥d]と同じ）
¥w	英数字（[A-Za-z_0-9]と同じ）
¥W	英数字以外（[^¥w]と同じ）
¥s	空白文字（半角スペース・タブ・改行。[¥t¥n¥r¥f]と同じ。全角スペースは含まれない）
¥S	空白文字以外（[^¥s]と同じ）
¥n	改行文字
¥r	復帰（MacintoshではLFを表し，それ以外ではCRを表す）
¥t	タブ文字
¥f	改ページ文字
¥0	ヌル文字
¥cA	制御文字（この場合，コントロールA）
¥ooo	8進数表現
¥xhh	16進数表現
●位置を表すもの	
^	行頭
$	行末
¥b	単語の区切り
¥B	単語の区切り以外
●繰り返しを表すもの	
x?	0個または1個の「x」
x*	0個以上の「x」
x+	1個以上の「x」
x{m,n}	m個以上n個以下の「x」
x{m,}	m個以上の「x」
x{m}	m個ちょうどの「x」
x{,n}	n個以下の「x」
●後方参照と特殊変数	
（文字列）	文字列をグループ化する
$1	( )で囲まれた最初の文字列
$2	( )で囲まれた2番目の文字列（$3，$4，$5…以下対応するグループの参照）
$+	最後に( )で囲まれた文字列
$`	マッチした部分より前
$&	マッチした部分
$'	マッチした部分より後
●その他	
（東京！大阪！名古屋）	論理和（東京か大阪か名古屋）
¥	メタキャラクタ（^$+?.*(){}[]|¥）を文字として扱いたいとき

先月のスクリプトから

さて，先月ご紹介した複数のファイルから特定の文字列を含む行を表示するスクリプトをもう一度見てみよう。

```
#!/usr/local/bin/perl
print "検索文字を入力してください¥n";
$find_words = <STDIN>;
chop($find_words);
while(<>){
   if (/$find_words/) {
      print "$ARGV $_";
   }
}
```

このスクリプトでは，検索文字列を指定する際に，単に特定の文字列だけでなく，正規表現を使用したパターンが含まれているかどうかも検索できるんである。ためしに「^¥t」と指定すれば，行頭がタブで始まる行が表示されるし，「1998¥.1?¥d¥.[1-3]?¥d」と指定すれば，複数のファイルから1998年の日付がついたレコードの一覧表を得ることもできる。わざわざExcelなどの重いソフトを起動する必要も減るかもしれない。

ちなみに1行目の#!/usr/local/bin/perlは一種のおまじないであるが，Mac環境以外では「パスを通す」というそれこそおまじないのような儀式があり，それらと互換を取れるようになっている。なくても動作そのものには影響はないが，後述するようなオプションを付加してスクリプトの性格付けをすることもある。まあ別にあえて異を唱える根拠もないので，付けておこう。

s演算子

テキスト処理で最も重要で使用頻度が高いのは，やはり検索置換だろう。s演算子はとても簡単な構文で検索置換を実現する。

```
s/検索文字列/置換文字列/ig;
```

なんだ，sedと同じじゃないかという読者もいるかもしれない。その通り。Perlは様々な言語のおいしい所をうまく利用している。

頭に「s」を付け，検索文字列と置換文字列を「/」で区切るだけである。最後の「/」の後に付いている「i」と「g」はオプションで，「i」を付けると検索時にアルファベットの大文字と小文字を区別しない。「g」を付けると1行中に検索対象があるかぎり置換を続けるという意味である。よく使うオプションなので，覚えておくと便利だろう。

sedなどに慣れているユーザーは行末の「;」の意味がわからないかもしれない。Perlではこのように文の最後に「;」を置いて複文を可能にしている。ひとつの文・コマンドが終わったら「;」を入れましょうという決めごとにすぎないので，あまり深く考えないように（笑）。

さっそく，このs演算子と正規表現を使って，便利なスクリプトを紹介することにしよう。

```
#!/usr/local/bin/perl -pi.bak
s/^¥r//;
```

さて，なにをするスクリプトだろうか？「行頭のLF（復帰）コードを削除する」という意味で，勘のいい読者ならもうお気づきだろう。

Windows（DOS）経由のテキストデータを受け取った場合，改行は「CRコード＋LFコード」で記録されているため，Mac上で開くと行頭に表示されないゴミ文字として「LFコード」が邪魔になることがある。これを除去する専用のユーティリティなどもあるようだが，たったこれだけで専用のユーティリティとまったく同じ動きをするんである。

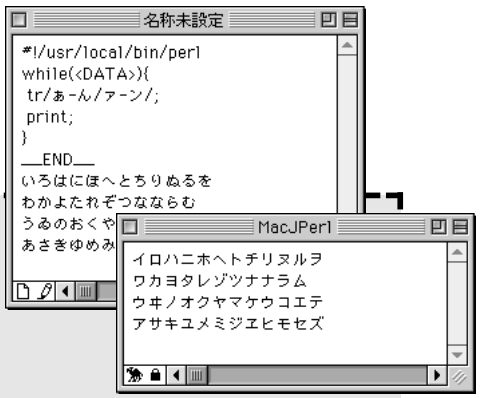
では逆に，MacからWindows環境へテキスト

MacPerl 入門

まかせて!! MacPerl

Tips

ちょっとテストしたいとき



スクリプトが長くなるような場合，ちょっと自信のない書き方に遭遇することがある。ちゃんと動くかどうかその場ですぐテストしたくなるような場面で，以下のようなテスト用のスクリプトがたいへん役に立つのでご紹介しておきたい。

```
#!/usr/local/bin/perl
while(<DATA>){
   #ここにスクリプトを書く
   print;
}
__END__
ここに処理したいテキストを入れる
```

このとおりタイプし終えたら，Command＋Shift＋Rを押せば，図のような結果がすぐに出てくるという仕組みだ。

これは<DATA>というファイルハンドルを開いて，__END__以下のテキストを1行ずつ読み込み，処理したものをSTDOUT（ウィンドウ上の標準出力）に出力している。

このほかにも，<DATA>ハンドルを使わなくても，スクリプトを開いた状態でCommand＋Shift＋Rを押せば，標準ウィンドウが開き，そこにテキストをタイプするかコピー&ペーストすれば結果が出力される。このときプログラムはずっと入力待ち状態にあるので，Control＋Dを押すことでスクリプトの実行を中止できる。

ファイルを持っていきたい場合はどうすればいいだろうか？

```
#!/usr/local/bin/perl -pi.bak
s/¥n/¥n¥r/;
```

これでWindows使いの方からのクレームがひとつ減った。口さがないWindowsユーザーからの「Macにはリターンキーがないらしい」というジョークも一蹴できるだろう。

1行目に「-pi.bak」という見慣れないオプションが付いてる。本来は出力ファイルのハンドルを開いた後，指定ファイルの各行をループさせてprintさせなければならないだが，この程度の簡単なものなら，これらの定型作業をオプションで処理させることができる。このスクリプトをドロップレットとして保存し，処理ファイルをドラッグアンドドロップすれば，オリジナルの処理ファイルには「.bak」という拡張子がつき，処理後のファイルが新規作成される。

後方参照

マッチした文字列を後で使えるようにする機能である。s演算子と組み合わせると大変便利な使い方ができる。なにはともあれ，具体的なスクリプトを見てみよう。

下記のスクリプトは今年の1月1日から実施された携帯電話・PHSの11桁化を自動化するスクリプトである。大阪06局の市内局番の4桁化も同時に行なえるようにした。アドレスデータの処理がまだ済んでいない人には便利だと思う。ただし，ハイフンの位置が違っていたり，パーレンを使っている場合などは手直しが必要であることをお断りしておきたい。

最初の「(¥D)?」は数字以外の文字が頭に付くことを条件にしている。「?」をつけるのは行頭からいきなり電話番号が始まる場合を想定しているからだ。番号の前に「携帯」とか「PHS」とか決まった文字列がくるのが分かっていれば「(携帯)」や「(PHS)」にすればより確実であろう。

これらの検索文字列のなかで「( )」に囲まれているグループを一番左から$1，$2，$3に対応させ，置換文字に反映させている。これらは次のマッチが行なわれるまでの局所変数である。つまり，マッチした文字列をこういった特殊変数に一時的に代入していると考えればいい。

$1が「${1}」となっているのは変数と次に続く文字列とを区切っているをとを意味する。$1に続く文字が数字でなければ，特に必要はないのだが。sedやAWKなどでは後方参照は$9（もしくは¥9）までしか使えない。Perlでは対応するグループさえあればどこまでも面倒を見てくれるんである。

1行中で「#」から改行まではコメントと見なされる。ほかにもコメントアウトさせる方法があるが，ほとんどこれで間に合ってしまう。Perlスクリプトは複雑になればなるほど後々のメンテナンスが大変になる傾向があるので，適宜コメントをつけるべきだ。半年後にはきっと自分自身に感謝するに違いない。

この他にも後方参照には，マッチした文字列全体を表す「$&」などがある。以下の例は，文になっている（行末が「。」で終わっている）のに行頭に全角スペースがない行に全角スペースを付け足すスクリプトである。

```
#!/usr/local/bin/perl -pi.bak
s/(¥D)?0([56])0¥-(¥d{3}¥-¥d{4})/${1}070¥-$2$3/;#PHS
s/(¥D)?0([1-48])0¥-(¥d{3}¥-¥d{4})/${1}090¥-$2$3/;#携帯電話
s/(¥D)?06¥-(¥d{3}¥-¥d{4})/${1}06¥-6$2/;#大阪06局内
```

筆者紹介
市川せうぞー
1964年9月12日生まれ，B型。
都内のDTP制作会社に勤務しつつ，自宅でSOHOを実践中。その結果，昼も夜も，休日も平日も，趣味も仕事も，愛妻も愛猫もすべての境目を失いつつある。
JAGAT（日本印刷技術協会）認定・DTPエキスパート。
http://www.asahi-net.or.jp/~ym3s-ickw/showtime.html

```
#!/usr/local/bin/perl -pi.bak
s/^[^ ].+。$/ $&/;
```

置換を式として評価する

s演算子でちょっと便利で変わったことができたりする。以下はあまりよい例とはいえないのだが。

```
#!/usr/local/bin/perl -pi.bak
s/¥d+/$& + 1/eg;
```

処理ファイル中の数字にすべて1を足している。たった1行のプログラムでこんなことも出来てしまうのだ。正規表現をちょっと工夫すれば，毎年年齢をひとつずつ上げなければならない会員名簿などでは重宝するかもしれない。

これは，s演算子の最後に「e」オプションを付けるだけで，それを式として評価しているという仕組みなんである。「e」オプションについては他にもいろいろな使い道があるが，おいおい説明することにしよう。ここでは置換が式として取り込まれ評価されていることがわかればそれでよい。

tr演算子

ある特定の文字と別の文字を1対1で置き換える演算子である。たとえば，全角のアルファベットと数字をすべて半角に置き換えたい時は，次のようにすればよい。

```
#!/usr/local/bin/perl -pi.bak
tr/A-Za-z0-9/A-Za-z0-9/;
```

データ入校の際にはこのような文字種の統一は必須であると思う。よほど組版に詳しいクライアントでないかぎり「6」と「６」，「A」と「Ａ」は同義なのだ（だから当然混在している）。にもかかわらず，制作会社によってはこういったテキストに無分別なところが少なからずある。ややもすれば，クライアント原稿絶対重視の偏見すら根強い。結果として，醜い（そしてシロートっぽい）組版が世間に出回ることになる。そもそも，データ入校そのものがそんな歴史のあることじゃないんだからして，文字コードに精通したプロフェッショナルの手を経なければ日本語の組版などゆめゆめ不可能なのであることを肝に命じたい。そういう美意識のかけらもないプロダクションは自然淘汰されていくからいいんであるが…。

つい愚痴ってしまった（自爆モード？　笑）。前出のスクリプトは何ら説明もいらないほど簡単なのであるが，ちょっと説明するなら「A-Z」はアルファベットの大文字26文字に展開された状態を意味している（ＡＢＣＤＥＦＧ……Ｚ）。Perlでは「/ /」に囲まれることによって，展開できるものは展開されているのだ。

tr演算子にもいくつかのオプションが存在するが，ほとんど使うこともないので，説明はしない。興味があるならLarry Wall他『プログラミングPerl』（オライリー・ジャパン）などを参照していただきたい。

Note ;

今回紙面で紹介したスクリプトはすべてCD-ROM MACLIFEに収録した。Perlの正規表現とパターンマッチがいかに簡単で強力なものかが，おわかかりいただけると思う。需要に合わせてカスタマイズできれば，あなたも立派なPerlプログラマーなんである。

さて，このような検索置換だけがPerlのすべてだろうか？　たしかにこれだけでも十分便利なんであるが，これだけのことなら冒頭の「Jedit」とAppleScriptを駆使すれば（恐ろしく遅いが）できなくはないだろう。コマンド数制限などがあるもののsedなどでも不可能ではない。

Perlは汎用言語であるからして，変数や制御構造を利用してより便利に仕事をこなすことができる。次回からはそれらについて説明しようと思う。お楽しみに（って楽しみにしてる読者なんているのだろうか？　ちょっと不安…）。

●やり方は何通りもあるってこと

あたしもかつてはまっすぐな眼をした純粋な少年だったが，いまではその瞳もすっかり淀みしょぼくれ，黒い腹はいまだ成長の一途をたどっている（笑）。

他人の不正が許せないまっすぐな方々には不幸なことだが，Perlはあらゆる人間性と嗜好に寛容な言語だといえる。他人の書いたプログラムを解析するのは容易なことではないし，それはそのまま自分の書いたものですらあてはまる。

実際に同じロジックの中でさえ書き方は何通りもできたりする。切り口を変えて考えればその選択肢は文学のような様相を呈するだろう（事実，Perl詩などという分野まである!?）。

じゃ「だれでもわかかりやすいように書きましょう」と一概にいえなかったりすることがあることも事実だ。

大きな迷路の中で，出口までの最短のコースを辿れば，最も速く，効率よく，燃費よくゴールすることができるだろう。それと同様に世界中のPerlハッカーは，より速く効率がよくメモリを浪費しないプログラムに執心している。そしてあなたもいずれその魅力の虜になるだろう。速さの魅力にはだれもかなわないもの。

しかしLarry先生は言っている。「ボスにクビにされるまえに仕事が終えられる。それがよい（正しい）プログラムなのだ」と。

あたしたちは言語学者ではない。どんなに美しく効率のいいプログラムも，納期や締め切りに間に合わなければ，単なるおバカさんに過ぎない。作家風情を気取っている場合じゃない。例外処理やメモリ管理で重箱の隅をつつくようなプログラムは，個人での使用が前提ならば問題の核ではないはずなんである。

それゆえ（そして幸いにも！），There's More Than One Way To Do It!（やり方は何通りもある）とはPerlの最も重要なスローガンなのだ。

There's More Than One Way To Do It!

MacPerl 入門

まかせて!! MacPerl

編集者・デザイナー・DTPオペレーターのためのMacPerl (3)

CD-ROM

市川せうぞー●有限会社シンクス

DTPの周辺には
様々なテキストファイルがあふれている。
使い捨てのデータを作るだけじゃなく
情報の抽出や再利用が出来なければ
真のデジタルの恩恵には
預かれないはずだ。
ワンソースマルチユースを実現するPerl。
その魅力とパワーを紹介するページ。
"All I want is Easy Action!!"

先月号でご紹介したパターンマッチだけでもPerlのすごさ・簡単さを十分理解していただけたと思う。しかし，Perlはあなたの想像以上に便利な道具なんである。

変数とか制御構造などと聞くだけで及び腰になってしまう読者もいるかもしれない。若かりし日，BASICなどでの挫折がトラウマになっていまだ心の痛む人もいるだろう。Perlは汎用言語であるからして，「変数」や「制御構造」を使えば複雑な処理をより簡単でパワフルに処理することが可能になるだろう。便利な道具を使いこなすには，それなりのテクニックだって必要になることがある。「便利」という言葉の誘惑に，困難を乗り越える力があることを信じて。

変数

「変数とは入れ物です」といっても，分からない人にはなかなか分かってもらえない。しかし，すでにもう第1回目の最初に掲載した文字数計算プログラムから変数を使っているのだ。そのまま紹介しても芸がないので，複数ファイルをドラッグ＆ドロップしても良いようにしてみよう。

```
#!/usr/local/bin/perl
while(<>) {
   $count += s/./$&/g;
   if (eof) {
      print "『$ARGV』の文字数は
$count文字です¥n";
   }
}
```

処理結果から，$count には文字数が入っていて，$ARGV にはフルパス名が入っているだろうと予測できる。そのとおり。このように変数は任意の情報を記憶したり，呼び出したりできる大変便利な仕組みなんである。

「$count」はユーザーが定義したスカラ変数で，「文字列」，「数値」，「真偽（ブール）値」のいづれかひとつを入れることができる。代入式は「$〈変数名〉=〈値〉;」で，変数の型はそのコンテキスト（文脈）で評価される。つまり，右辺値が数値を返すとき，その変数は数値型であり，文字列を返せば文字列型になる。Perlはつくづく都合よく作られているんだなぁと感心してしまう。

前記の例で説明すると，s/./$&/g は置換に成功した回数（数値）を返す。これを単純に代入するのではなく，「+=」で加算している。つまりこの場合，$count = $count + s/./$&/g; と同義になるということだ。

明示的に文字列を代入したい時，一組の「'」シングルクォートで囲む必要が生じる。例えば，数字「5」を $str に文字列として代入したい場合は $str = '5'; とする。右辺値が裸の文字列の時も同様に $str = '文字列'; としなければならない。また，5行目のprint文と同様に文字列中の変数を展開して代入したい時は，一組の「"」ダブルクォートで囲めばよい。もちろん，変数に変数を代入することだって可能だ。

スカラ変数については，MACLIFE CD-ROMのサンプルスクリプトフォルダに「煙草税の計算」を試作してみたので，各々の変数が何を保持しているか眺めてみるといいかもしれない（Perlの変数が政治家の理不尽な施策よりも一貫性を持っている証として :-）。

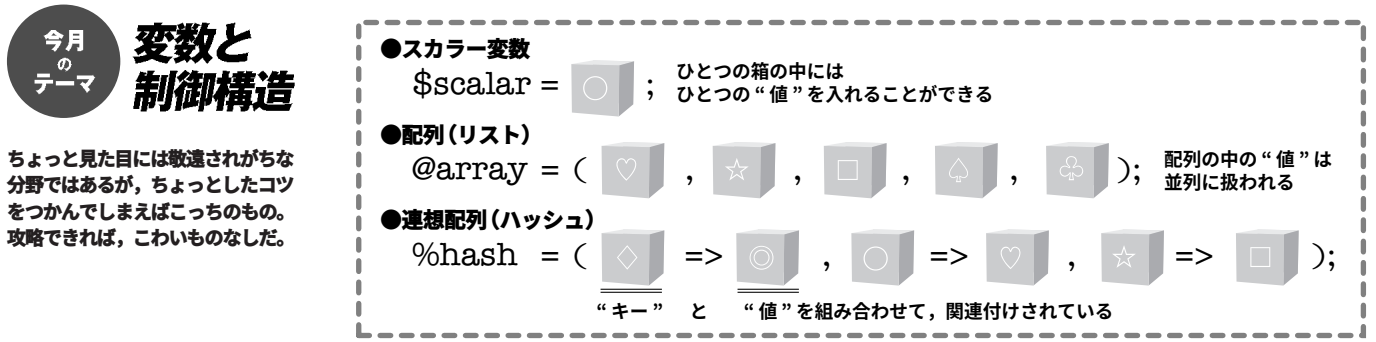
配列（リスト）・連想配列（ハッシュ）

ときおり，一度にたくさんのデータを同じ次元で扱えると幸せなことがある。大量にスカラ変数を作ってプログラムを汚す必要はない。

例えば，QuarkXPressのタグ処理で，ビブロスフォント（GthinHroundの丸数字）を指定したい時は，以下のようにするといいだろう。

今月のテーマ **変数と制御構造**

ちょっと見た目には敬遠されがちな分野ではあるが，ちょっとしたコツをつかんでしまえばこっちのもの。攻略できれば，こわいものなしだ。



変数の周辺事情

```
#!/usr/local/bin/perl -pi.bak
@maru_suji = ('p', 'q', 'w', 'e', ➡
'r', 't', 'y', 'u', 'i', 'o');
s/●(¥d)●/ ➡
<¥$f¥"GthinHround¥"> ➡
$maru_suji[$1]<¥$f¥$>/g;
```

注）スクリプト内の➡印は，改行せずに下の行が続くというマークです。

配列は「@」から始まる名前を持つ。この場合 @maru_suji は「'p', 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o'」の10個の要素を持っている。右辺値が「,」で区切られた裸の値なら，演算子の優先順位から全体を「( )」で囲う必要がある。

配列の各々の要素の呼び出すためには，「$〈配列名〉[〈添字〉]」で，添字は0から始まる整数で指定する。右辺値の最も左から0番…1番…2番…となる。$maru_suji[0] は「p」を返し，$maru_suji[5] は「t」を返す。配列の最後の添字には $#〈配列名〉というスカラ変数を使用でき，$maru_suji[$#maru_suji] とすれば「o」を返す。配列の要素数が不定の場合などに便利だ。

配列が便利だと言っても，欲しい情報が常に整数順に並んでいるとは限らない（世界は無秩序で順序よくナンバーリングできほど単純じゃないことの方が圧倒的に多いはず）。ビブロスフォントの例を挙げるなら，PifontWk1の「日～土」をタグ処理するためには，'○日○' が 'h' と関連づけられなければならない。添字に「○日○」が使えれば問題は一挙に解決する。

```
#!/usr/local/bin/perl -pi.bak
%youbi = (
'○日○' => 'h',
'○月○' => 'i',
'○火○' => 'j',
'○水○' => 'k',
'○木○' => 'l',
'○金○' => 'm',
'○土○' => 'n');
```

●変数名の付け方

変数名の付け方には簡単な法則がある。それぞれの記号の次の2文字目はアルファベットか「_」が使えるが「_」を使った場合，3文字目が必要になる。3文字目以降は任意の長さでアルファベットと「_」と数字を使って命名できる。

アルファベットの大文字と小文字は区別され，全く別のネームスペースを持ち，違う変数として認識される。$count と $COUNT と $Count はそれぞれ違う値を持った別々の変数である。しかし，できるというだけで，実際に大文字と小文字で変数を区別するなんてことはまずしないだろう。むしろ，陥りやすいミスとしての注意が必要だ。

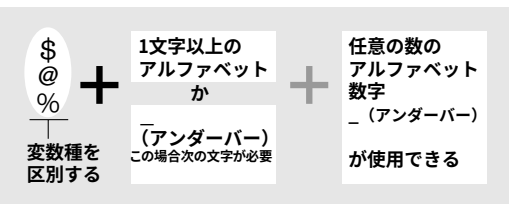
変数名の長さには制限はないが，できるだけ簡潔に，かつ分かり易い名前が良い。後々のメンテナンスを考えれば言うまでもないだろう。個人的には8文字～10文字程度の具体的な名前と，もし同様な扱いをしたいグループがあるなら「 _」で区切ってグループ名か追番をつけるようにしている。まあこれはあくまで趣味の問題なので…。

ちなみに，リストやハッシュの各要素を呼び出す場合，$〈変数名〉に〈添字〉のように「$」を付けるわけだが，呼び出された値そのものはスカラなので「$」が頭に付くということを忘れてはならない。

●Perlにとっての変数

local(〈変数〉)関数を使ってローカル変数定義をしないかぎり，ほとんどの変数はグローバル変数である。つまり，一度定義すればスクリプトのどこからでも参照できるようになる。これはPerlにとって美徳だが，人間を必要以上に無精にしてしまう欠点を持っている。あちこちで参照したり，変更したりしているうちに，思わぬ落とし穴が待っているかもしれない。変数に変な（予想外の）数が入ってしまったら，シャレにならない。

Perlで使われる変数は改まった型宣言や初期化をする必要はない。どこでもいきなり使うことができ，初期値には未定義値がセットされている。しかしこれも「全体の見通し」という意味で言えば，ちょっと考えものかもしれない。短いスクリプトならそれでも構わないのだが，長いスクリプトを組むような場合ではスクリプトの最初で初期化をし，コメントを付けることをオススメしたい。見通しが格段によくなることは請け合いだ。




```
s/○[日月火水木金土]○/ ➡
<¥$f¥"PifontWk1¥">$youbi{$&} ➡
<¥$f¥$>/g;
```

連想配列は「%」から始まる名前を持っている。「'○日○' => 'h'」はキー（添字）と値という関係を持ち，読み込み順も要素順も一切考慮されない（まさに無秩序に）。「=>」は「,」を用いて，; までをすべて1行で書くこともできるのだが，キーと値の関係を見やすくするためにこのような書き方が好まれる（先月号のコラム“There's More ThanOne Way To Do It!”を思い出して）。

連想配列の各々の値の呼び出しは「$〈連想配列名〉{〈キー〉}」で，$youbi{'○月○'} は「i」を $youbi{'○土○'} は「n」を返す。

連想配列を使って半角カナを全角カタカナに変換するスクリプトをCD-ROMに添付したので参照していただきたい。

特殊変数

Perlによってすでに予約された変数というのが存在している。前記のスクリプトの例でいえば $ARGV がそれにあたる。$ARGV は現在処理中のカレントパス名を保持している特殊変数だ。

Perlで最も頻繁に登場する特殊変数といえば「$_」だろう。カレント処理行を保持しているのだが，明示的に使われることはあまりない。たとえば，ただ単に「print;」と書けば「print STDOUT $_;」を意味し，標準出力ウインドウにカレント行をprintする。「s/xx/yy/;」と書いた場合も，カレント処理行（ $_）が対象であり「$_ =~ s/xx/yy/;」を意味している。

特殊変数は便利な（そしてなくてはならない）存在だが，ユーザーが不意に変更を加えてしまう可能性もある。もし，そのような心配があるなら「#!/usr/local/bin/perl -w」などと-wオプションを付けることで，予約語との衝突や誤った使い方をしている場合に警告を発するようになる。

MacPerl 入門

まかせて!! MacPerl

制御構造

「制御構造」などと難しい漢字を使って書いてみたが，要はプログラムにいろいろな条件を与えて，ちょっと賢い動きをさせる仕組みだと思っていただければよい。

もし「初めて上京してきた両親に池袋で降りてそこから電話を掛けて欲しい」というシチュエーションをPerlで書くとしたら，CD-ROMに添付されている「池袋で電話をかけて！」というようになる（もちろん，これで本当に電話が掛けられるといういうものではないのだが(^^;)。

制御構造を簡単に書くと「〈制御種類〉(〈条件式〉){〈実行ブロック〉}」というカタチになる。制御種類の差こそあれ，条件が“真”の時，実行ブロックが実行されることには変わりはない。

ここで，Perlにとって何が“真（True）”でなにか“偽（False）”なのかをはっきりなければならない。

答えは簡単。コンテキストが「"" (空文字列)」か「'' (文字列)」か「0（数値）」か「未定義値」を返せば，これらは“偽”である。そして，これら以外はすべて“真”なのだ。さらにPerlではほとんどすべてのものを真偽値を得るために評価できる。

if とunless（条件分岐）

if文の構文は以下のとおり。

```
if (〈条件式1〉) {
   〈条件式1〉が真の時このブロックを実行
} elsif (〈条件式2〉) {
   〈条件式2〉が真の時このブロックを実行
} elsif (〈条件式3〉) {
   〈条件式3〉が真の時このブロックを実行
} else {
   〈条件式1～3〉がすべて偽だった時
   このブロックを実行
}
```

elsif節とelse節は無くてもかまわないのだが，最初の〈条件式1〉以外の処理をさせたい時，便利な書き方だ。必要であれば，elsif節はこのようにいくつも設定できる。

逆に，単純なことだけをやりたいときは「〈やりたいこと〉if〈条件〉;」と書くこともできる。

サンプルスクリプトとして「mm-Q-point」換算を例にして，CD-ROMに添付した。

〈条件式〉が偽の時だけ処理を行いたいことがあるかもしれない。「if (!〈条件式〉) {ブロック}」のように条件に否定演算子「!」を付けて書いてもいいのだが，「unless (〈条件式〉) {ブロック}」の方がスマートだと言える。なぜなら，やりたいことはいつも“真”である確率が高いからだ。

whileループとuntilループ（繰り返し1）

物事を繰り返すことを人間は最も不得手としている。そして，コンピュータが最も得意とする分野がこの繰り返し処理だ。perl は様々な繰り返し方を知っている。

```
while(〈条件式〉) {
   〈条件式〉が真である間このブロックを
繰り返す
}
```

これまで何度も出てきた「while(<>) {ブロック}」は処理ファイルを読み込む常套句であるが，これを省略なしに書くと「while($_ = <ARGV>) {ブロック}」と書くことが出来る。「<ARGV>」はカレントファイルが開かれた状態の特殊なファイルハンドルで，「<>」と省略しても構わない。そこから$_（カレント処理行）へ1行づつ代入しているのだ。そしていうまでもなく $_ も省略可能であることから「(<>)」はカレントファイルから1行ずつ読み込むという動作をする。

もし読み込みファイルの中に空行があったらどうなるだろうか？　空行はなにもないから空文字列（偽）が返ってループを抜けてしまうのだろうか？　実際にはそうならない。なぜなら，空行には改行文字「¥n」が残っており，（れっきとした）“真”が返るからだ。

untilループはちょうどif文に対してのunless文と同様に条件が“偽”のときにブロック内をループする。

forループ（繰り返し2）

forループはwhileループと似た動きをするのだが，ちょっと外見が違う。簡単なテストをしてみよう。

```
for ($i = 0; $i < 10; $i++) {
   print "$i ¥n";
}
print "ループ終了¥n";
```

forに続く条件式は「;」で3つに区切られている。1番目の式（$i = 0）は初期条件の設定し，2番目の式（$i < 10）で条件が評価される。もし“真”であればブロック内（print "$i ¥n";）が実行される。ブロック内を実行し終わってループの先頭に戻った時，3番目の式（$i++）で初期条件が更新されて，また2番目の式に当てはめて再評価される。“真”であればブロックが実行され，“偽”であればループを抜けられる。

最初にforループはwhileループと似た動きをすると書いたが，実際にwhileループで書くと以下のようになるだろう。

```
$i = 0;
while ($i < 10) {
   print "$i ¥n";
} continue {$i++}#$iをインクリメント
print "ループ終了¥n";
```

foreachループ（繰り返し3）

foreachループは配列の各要素を拾いながら実行ブロック内をループし，すべての要素が終わるまでひとつづつスカラー変数に代入していく。

```
foreach〈変数〉(〈配列〉) {ブロック}
```

この時の変数はforeachループの中だけのローカル変数で，省略すると「$_」が使われる。

前出の「池袋で電話をかけて！」がforeachループの簡単なサンプルになっているので，参照していただきたい。

ループ制御

人間だって物事の途中で気が変わることはたびたびあるものだ。制御構造の実行ブロック内で，制御の流れを自在にコントロールすることができれば，プログラムに余計なことをさせる必要もなくなり，デバッグも格段にラクになる。

これは，last文・next文・redo文の簡単なテストだが，それぞれの振る舞いがよくわかるように，入力された文字列がそれぞれのコマンドを含んでいたらそのコマンドを実行するようにし，さらに，ループの入り口と出口で評価対象のスカラー値（$i）を表示するようにした。

```
for($i = 0;$i<10;$i++) {
  print "▼Loop Start!  ¥$i = $i, ➡
¥$_ = $_¥n";
  $_ = <STDIN>;#キーボードからの入力
  last if /last/;#'last'と入力されたら
  print "through of last… ¥n";
  next if /next/;#'next'と入力されたら
  print "through of next… ¥n";
  redo if /redo/;#'redo'と入力されたら
  print "through of redo… ¥n";
}
print "▲Loop End! ¥$i = $i, ➡
¥$_ = $_";
```

last文は強制的に実行ブロックを抜け，ループの外に処理を移す。next文はブロック内の残りの処理をとばしてもう一度条件式を評価し，ブロック内に入るべきかどうかを判断する。redo文は条件式を評価することなくもう一度ブロックの最初から実行しなおしているのがお分かりいただけると思う。

Note;

さて，変数と制御構造について急ぎ足で紹介してきたが，さすがにこの連載もおちゃらけだけでは済まなくなってきてしまった（笑）。

こんなことが何の役に立つのか不安になってきた読者もいるかもしれない。もちろん，スクリプトを設計できるようになるのがベストなんだが，他人によって書かれたものを読み解くためには，こういう知識は不可欠なのだ。これらの約束事をマスターすれば，（世界中の）膨大なPerlスクリプトを自分の目的に合わせてカスタマイズすることができるようになる。そして幸運なことに，やりたいことのほとんどは既に誰かの手によって書かれており，ちょっと手直ししてやれば，目的はより速く達成するだろう。

ファイルハンドル

●ファイルハンドルとはなにか

ファイルハンドルとはPerlの内部でデータの入出力を制御する部分である。入出力と一言で言っても，種類はさまざまで，「STDIN」「STDOUT」「STDERR」などのPerl標準定義の入出力（すべてウィンドウ上）から，ファイル操作，プリンタなどのデバイス，ネットワークに対するソケットなどユーザーが定義する入出力操作を行うためにファイルハンドルが存在する。

すべてのファイルハンドルの説明を限られた紙面ですることはできないので，もっともよく使われるファイルへのアクセスについて説明することにしよう。

●ファイルへの出力

先月号で紹介したDOSファイルのLFコードを取るスクリプトを「-pi.bak」オプションを使わないで，ファイルハンドルを使って書き直してみよう。全く同じではつまらないので，今度はオリジナルファイルのファイル名はそのままで，処理後のファイルに「.new」という拡張子を付けるようする。

```
#!/usr/local/bin/perl
while(<>){
   open(OUTPUT, ➡
">$ARGV.new") ➡
unless -e "$ARGV.new";
   s/^¥r//;
   print OUTPUT;#出力
}
close OUTPUT;
```

標準の「STDIN」から入力するためには，「$_ = <STDIN>;」などと書けばよいのだが，ファイル出力用のユーザー定義のファイルハンドルを開くためには，「open(〈ファイルハンドル名〉, ">〈フルパス名〉");」としなければならない。これで出力ファイルとPerlのファイルハンドルが結びつけられたわけである。フルパス名（この場合はカレントファイルを保持している特殊変数$ARGVに“.new”を付けている）の前に「>」を付けることでこのファイルハンドルが出力用にオープンされていることを示している。もし，すでにあるファイルに追加して書き込みたい場合には「>>」とすればよい。

「unless -e "$ARGV.new"」はボリュームがロックされていたり，同名のファイル名や長すぎるファイル名になってしまうような時にそのファイルの処理を飛ばすようになっている。常にファイルの書き込みが許可されているとは限らないのだ。

実際にファイルに書き込む場合も，「print OUTPUT;」のようにファイルハンドルOUTPUTに対して書き込んでいる。

一番最後に「close OUTPUT;」としているのは開いたファイルハンドルを明示的にクローズしている。スクリプトが終了するか，再びopen関数が呼び出されると現在開いているファイルハンドルは，暗黙の内にクローズされるが，行番号を保持している特殊変数$.がリセットされない。「開けたドアはちゃんと閉めなさい」というお母さんの小言を思い出してしまう（笑）。

●ファイルからの読み込み

MacPerlでは普通「while(<>)」と書けば，ドラッグ＆ドロップされたファイルを読み込む仕組みなので，特に必要は感じないのだが，特定のファイルを参照したい場合などは以下のようにすればよい。

```
open(〈ファイルハンドル名〉,➡
"〈フルパス名〉") || die ➡
"Can't open : $!¥n";
```

〈フルパス名〉をファイル名だけにすると，スクリプトの置かれた階層だけに限定される。開かれたファイルハンドルから1行づつ読み込むためには「while (<〈ファイルハンドル名〉>) {ブロック}」とすればよい。

「|| die "Can't open : $!¥n"」は読み込みファイルのオープンに失敗したらスクリプトをdie関数で中断させるという意味だ。「$!」はシステムエラーの内容を保持する特殊変数で，die以降はすべて標準エラー出力であるSTDERR（ウインドウ上）に出力される。

これは余談だが，ユーザー定義のファイルハンドル名には大文字を使うことをオススメしたい。予約語と衝突することもないし，なんだがいかにもファイルハンドルっぽいからなんだが。

MacPerl 入門

まかせて!! MacPerl

編集者・デザイナー・DTPオペレーターのためのMacPerl (4)

CD-ROM

市川せうぞー●株式会社シンクス

**DTPの周辺には
多様なテキストファイルがあふれている。
使い捨てのデータを作るだけじゃなく
情報の抽出や再利用ができなければ
真のデジタルの恩恵には
あずかれないはずだ。
ワンソースマルチユースを実現するPerl。
その魅力とパワーを紹介するページ。
*"All I want is Easy Action!!"***

Perlが便利だという理由のひとつに，関数が豊富であることが挙げられる。実際，日常的な処理の多くは関数を使えばカンタンに実現することが多い。今回は，配列関数の説明を中心に，そのエッセンスを感じてほしい。

あぁ，しびれるほど便利！

生成と分離 (配列関数1)

配列それ自体は入れ物にすぎないが，関数を使えば，配列を自由に操作することができる。実は配列は，関数を使うことで，初めてその真価を発揮するんである。

まず，スカラーから配列を生成してみよう。怠け者のPerlにとって，配列は記述し代入するものじゃなく，生成するものだったりする。目からウロコが…。

split(/〈区切り文字〉/, 〈スカラー〉, 〈制限〉)

split関数は区切り文字（デリミタ，省略すると空白文字）によってスカラーを分割し，配列を生成する。区切り文字は「/ /」で囲まれていることでもわかるとおり，正規表現も使えるし，複数の文字列もOKだ。スカラーを省略すると，ご存じ「$_」(カレント処理行)が適用される。制限を付ければ生成される配列の個数を決められる。たとえば，確実に連想配列を生成するためには，

```
$str = 'Japan Keizo Obuchi';
($country,$name)=split(//,$str,2);
$president{$country}= $name;
print "$president{$country}¥n";
```

としたりする。この場合，ひとつ目のスペースだけで分割され，配列は2つしか作られない。ひとつめのスペース以降は，すべて2つめの配列要素に含まれてしまう。

離れたりくっついたりするのは，芸能界ばかりじゃない。split関数でスカラを分離したかと思えば，配列の各要素をくっつける関数だってPerlにはちゃんと用意されている。

join (〈デリミタ〉, 〈配列〉)

まあ，単純に配列の各要素をデリミタでつないで文字列として返すだけなんだが，これは例を見てもらうほうが理解が速い。

```
$str = 'A☆B☆C☆D☆E☆F';
@array = split(/☆/, $str);
$str = join('●', @array);
print "$str";
```

明示的に「$str」に代入するまでもなく，単に結果をprintするだけなら「print join('●', @array);」でもいい。$strは，まずsplit関数によって@arrayを生成する。このときデリミタは「☆」になっているから，@arrayは「('A', 'B', 'C', 'D', 'E', 'F')」になっている。これをjoin関数を使ってひと続きの文字列にする。このときのデリミタは「●」なので，配列の各要素の間に「●」が挟まれているということだ。

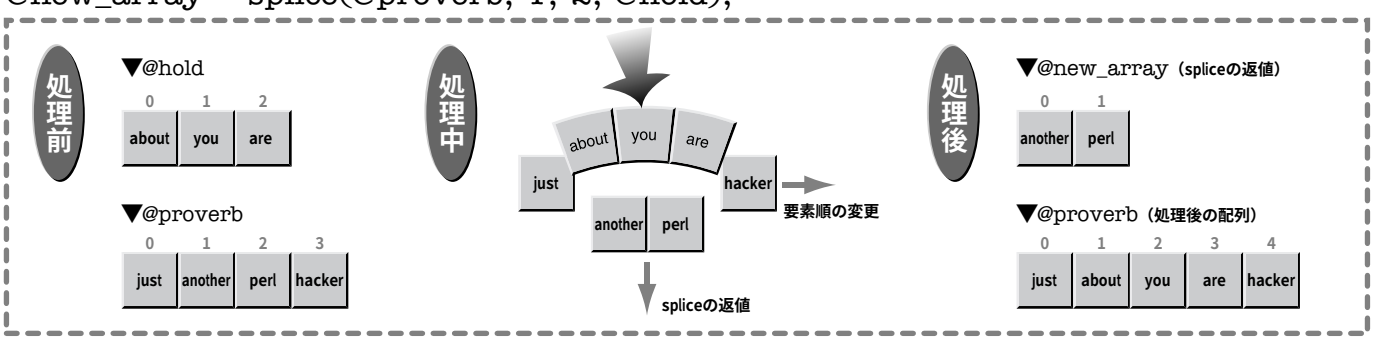
基礎的な説明だけではつまらないという読者のために，MACLIFE CD-ROMに「表入力_縦方向にする」を添付した。タブ区切りでヨコ方向に入力されている表組み用テキストを，そのコラムごとに縦方向に置き換えるものだ。これを使えば，面倒で退屈なコピー＆ペーストから解放されるはず。

今月のテーマ

関数

複雑な処理をカンタンに実現してくれる関数。この豊富な関数群がなかったら，きっとPerlの魅力は半減してしまうだろう。

```
@new_array = splice(@proverb, 1, 2, @hold);
```



切った張ったは…？（配列関数2）

政治家がおイタをしでかした時，決まって秘書のクビを切って身の潔白を証明するという常套手段がある。こんな便利な方法をPerlが放っておくはずがないではないか？

```
@tokage = ('センセイ','家族','秘書');
$shippo = pop(@tokage);
print "$shippoがやったことで¥n",
join('と', @tokage), "は知らなかっ➡
たんですぅ";
```

●●●●●●●●●●●●●●●●●●●●●●●●●

```
秘書がやったことで
センセイと家族は知らなかったんですぅ
```

注）スクリプト内の➡印は，改行せずに下の行が続くというマークです。

まさにトカゲの尻尾切りとはこのことだろう。pop関数は，配列の最後の要素を取り出して，配列全体のサイズをひとつ小さくする。身軽になれたトカゲはうまく逃げ切れたのだろうか？

いやいや，このセンセイひとりじゃ何も出来ないんで，仕方なく新しい秘書を雇うことになった。しかし，いつまた不正がばれてもすぐに代わり身が利くように配列（@tokage）の一番うしろに順序づける必要がある。

```
@tokage=('センセイ','秘書1','秘書2');
push(@tokage, '秘書3', '秘書4');
print join('と', @tokage);
```

●●●●●●●●●●●●●●●●●●●●●●●●●

```
センセイと秘書1と秘書2と秘書3と秘書4
```

こうして，新人の秘書3，秘書4はpush関数で@tokage配列のしっぽとなった。最後に雇われた秘書4にとっては，いつ自分が濡れ衣を着せられてしっぽ切りにあうか，わかったものではない。けれど,秘書4は非常に有能で,Perlに詳しいのであった。そしてクーデターは実行された。

```
@tokage = ('センセイ', '秘書1', ➡
'秘書2', '秘書3', '秘書4');
$atama = shift(@tokage);
unshift(@tokage, '有能な先生');
print "$atama さようなら…¥n";
print join('と', @tokage);
```

●●●●●●●●●●●●●●●●●●●●●●●●●

```
センセイ さようなら…
有能な先生と秘書1と秘書2と秘書3と秘書4
```

「$tokage[0] = '有能な先生';」なら低コストで同じことができるが，この場合，shift関数という政治手段が必要なのだから仕方がない。さらにunshift関数という政治手段で，有能な先生を迎え入れることに成功した。

次に着手すべきは組織改革だろう。ろくに仕事ができない秘書1と秘書2はSplice関数で外へ追いやって，第1秘書には自分がなるべきであると秘書4は考えた。

```
@tokage = ('有能な先生', '秘書1', ➡
'秘書2', '秘書3', '秘書4');
$shippo = pop(@tokage);
@kaiko = splice(@tokage, 1, 2, ➡
$shippo);
print join('と', @kaiko), ➡
"はさようなら…¥n";
print join('と', @tokage);
```

●●●●●●●●●●●●●●●●●●●●●●●●●

```
秘書1と秘書2はさようなら…
有能な先生と秘書4と秘書3
```

ちょっと寓話が長すぎて，DTPとのなんの関係もなくなってしまった（笑）。しかし秘書4もこれでしばらく安泰なんだろう。

splice（〈配列〉,〈削除開始要素〉,〈削除要素個数〉,〈挿入リスト〉）

splice関数の実際の動作は，左ページの図を参照していただきたい。

splice関数は引数をたくさん取るので，ちょっと見た目も悪く，敬遠されがちなのだが，配列の構造さえわかっていれば万能なんである（CD-ROMにサンプルを添付）。

ソート（配列関数3）

物事がきちんと整列していることは，人間にとってとても都合がよい。

ちょっと想像してほしい。広辞苑や電話帳が50音順でなかったとしたら，TVの番組欄が時間順でなかったとしたら，本の索引がアルファベット順でなかったとしたら，目次がノンブル順でなかったとしたら……？

まさにわたしたちは，整理されナンバリングされたインデックスを手がかりに世界を理解しているといっても過言ではない。

```
@array = ('かん', 'わたなべ', ➡
'なかじま', 'いちかわ');
@array_sort = sort(@array);
print join("¥n", @array_sort);
```

sort関数は指定された配列を昇順に並び替えた配列を返す。この場合，配列の各要素は文字コード順で並び替えられるため，結果的に仮名文字列はほぼ50音順になる。

文字コード順で並び替えられるということは，数字を数字として認識しないということにほかならない。ナンバリングされたファイル名がフォルダのリスト表示で破綻をきたすのと同じだ。しかしPerlには，数字の大小で並び替える方法もちゃんと用意してあるので，ご心配なく。

```
@array = (2,-9, 10, 0, 152, -256);
@array_sort = sort({$a <=> $b} @array);
print join("¥n", @array_sort);
```

{$a <=> $b}は特殊なサブルーチンの一種で，数値の大小を比較している。数値を降順に並び替える必要があるなら「sort({$b <=> $a} @array);」というように，「$a」と「$b」を入れ替えればよい。

「$a」と「$b」はブロック内（ローカル）の特殊変数なので変数名を変更してはならない。

ここでひとつ疑問が浮かんできた。文字列を降順にするためにはどうすればいいのだろうか。方法は2つある。ひとつはやはりこの特殊なサブルーチンを応用して使う方法で，「sort({$b cmp $a} @array);」という方法（cmpは文字列同士を比較する演算子）。もうひとつは，reverse関数を使う方法だ。

```
@array = ('かん', 'わたなべ', ➡
  'なかじま', 'いちかわ');
@array_sort = reverse sort(@array);
print join("¥n", @array_sort);
```

reverse関数は配列の要素順を逆転させる一種の並び替えといってよい。

この場合，いったん昇順にソートしたのち，その配列順を逆転しているから，結果的に降順にソートしていることになる。

ソートについてはそれ単体で価値のある関数なので，これだけで有用なアプリケーションができてしまう。CD-ROMに「Sort.pl」を添付した。任意のデリミタで区切られたフィールドで，文字コード順・数値順・いろは順・長さ順の昇順と降順ができるようにした。

MacPerl 入門

まかせて!! MacPerl

その他の便利な関数

関数の動きがだいたい理解できたところで，ここはひとつ便利そうな関数について「つまみぐい」をすることにしよう。

Perlを使うためにPerlのすべてを知る必要はない。おいしいところを「つまみぐい」するだけで，Perlは充分おいしいんである。

●標準入力（STDIN）から改行文字を取り除きたい

```
chop($ans = <STDIN>);
```

chop関数は最後の文字を取り除くのに使われる。ユーザーからの入力（STDIN）は通常，改行文字（¥n）を含んでいて，それを削除するため多用されるテクニックである。これを，もし配列コンテキストで使えばどうなるだろうか？

```
@array = ('あいう','abc','0123');
chop (@array);
print"@array";
```

なんと，各要素から最後の文字が削除されてしまった。

Perl5から，最後の文字列が「¥n」であるときだけ，これを削除してくれる「chomp」という関数が導入された。chomp関数を使えば，最後の文字が改行かどうかということを気にする必要がなくなる。

また，標準入力から最初の1バイトだけを読み込めばよいような場合であれば，getc関数も便利かもしれない。

```
$str = 'MACLIFE';
chomp($atr);#削除されない
print "$strはよい雑誌か？  [y/n]¥n";
$ans = getc;
print "$ans¥n";
```

●70バイトごとに入っている改行をつなげたい

```
#!/usr/local/bin/perl -pi.bak
chomp if length($_) >= 70;
```

ワープロなどできれいにインデントされている（ご親切に改行で揃えられている）原稿も，これで各行がつながってくれる。電子Mailや通信のログもこうして変換すれば，再利用が手軽になるというものだ。

length関数は引数のバイト数を返す。文字数でないことに注意してほしい。

●現在の時刻を知りたい

```
($sec,$min,$hour,$mday,$mon,➡
$year, $wday) = localtime(time);
$month = ➡
('睦月','如月','弥生','卯月','皐月', ➡
'水無月', '文月', '葉月', '長月', ➡
'神無月', '霜月', '師走')[$mon];
$week = ('日', '月', '火', '水', ➡
'木', '金', '土')[$wday];
$year = $year + 1900;
#2000年には100が返るから大丈夫
print "$year年 $month $mday日 ➡
($week曜日) $hour時 $min分 ➡
$sec秒¥n";
```

ちょっと見にくくて申しわけない。localtime(time)は現在の時刻の9つの要素（ちなみに上記以外に元旦からの通算日と夏時間の判定）を返す。値はすべて数字で，$monは（添字参照を考慮して）0〜11になっているので，注意が必要になる（実際の月と合わせるためには，+1すればよい）。

またPerlの標準ライブラリには，localtime関数と逆の働きをするtimelocal.plが含まれている。

●「4」や「10」を「004」や「010」と変換したい

```
foreach $i (1 .. 10) {
  $n = sprintf("%0.3d¥n", $i);
  print "$n";}
```

sprintfは「sprintf("〈フォーマット〉", 〈引数〉)」という形で，引数をフォーマットした値で返す。この要領でさまざまなフォーマット変換を可能にしている。「printf」と書けば，ファイルハンドルに直接プリントできる。ちなみに「1 .. 10」という書き方は，範囲演算子「..」を使って1から10までの配列を意味している。

```
foreach $n (0 .. 255){
  printf "%.2X : %c¥n", $n, $n;}
```

なんと，たったこれだけで1バイトのコード表になってしまう。

●一度取り込んだ値を式として再評価する

```
$str_1 = 'せうぞー';
$str_2 = 'print $str_1';
eval $str_2;
```

この意味がなんだかよくわからない人は，CD-ROMに添付した「One Liner」を参照していただきたい。入力した文字列がそのまま式として評価されることで，複数のファイルに対して，いちいちスクリプトを書くことなく，その式が適用されるという仕組みだ。いわゆるコマンドラインからのワンライナーに近い動きをする。

任意の値を再評価できるevalは，想像以上に可能性を持っているかもしれない。

●連想配列（ハッシュ）のキーのみ，値のみを取り出したい

```
%week = (
   'Mon.', '月', 'Tue.', '火',
   'Wed.', '水', 'Thu.', '木',
   'Fri.', '金', 'Sat.', '土',
   'Sun.', '日');
@english = keys(%week);
@japanese = values(%week);
print "@english¥n@japanese";
```

ループを使ってひとつひとつ取り出さなくても，このようにすればよい。順序よくソートしたい場合は，「@english = sort keys(%week);」とすればいいだろう。

●配列の中から，条件を含んでいるものだけを取り出したい

```
@pets = ("たま","みけ","ラッシー",➡
"とら", "パトラッシュ");
@meiken = grep (/[ァ-ン]+/, @pets);
$count = grep (/[ァ-ン]+/, @pets);
print "@meiken¥n計$count匹";
```

これも，配列をループに入れなくても，このように簡単に取り出すことができる。さらにこのように，これら関数はスカラコンテキスト（左辺値がスカラ）でその数値を返すことが多いことも覚えておくと便利だ。

筆者紹介

市川せうぞー

1964年9月12日生まれ，B型。
長年のオペレーター生活のため，ひどい腰痛に悩める毎日。所沢周辺で腕のいい鍼灸師を募集中。
JAGAT（日本印刷技術協会）認定・DTPエキスパート。
http://www.asahi-net.or.jp/~ym3s-ickw/showtime.html

information

鎌田幸雄氏を中心としてASUG（AppleScript User's Group；アップル公認ユーザーグループ）が発足し，活動を開始した。DTPユーザーも多く，メーリングリストも活発にやりとりされている。現在リッチなパトロンを募集中。
詳しくは以下のURLから。
http://www.abc.ne.jp/~ASUG/

MacJPerlがバージョンアップ（5.2.0r4-J1）され，細かなバグがフィックスされた。CD-ROM MACLIFEに収録。

サブルーチンを使う

●サブルーチンとは

プログラム中で，同じルーチンを何度も繰り返し使いたい場面があるかもしれない。しかし，何度も同じことを書くのは苦痛だし，見た目も悪い。第一，Perlの流儀に反している。

そこでPerlでは，そのようなひとまとまりのブロックをモジュール化して，プログラム中のどこからでも呼び出すことが可能になっており，それをサブルーチンと呼んでいる。

```
$num = 1234567890;
$ans=&commas($num);
print "$ans¥n";

sub commas {
   local($_) = @_;
   1 while ➡
s/(.*¥d)(¥d{3})/$1,$2/;
   $_;
}
```

これは，整数の位どりにコンマを入れるサブルーチンである。サブルーチン自体はどこの場所に書かれていてもよい。

呼び出す時は，どこからでも，&commas($num)という具合に「&〈サブルーチン名〉(引数)」とすることで呼び出され，その結果を返す。

サブルーチンに渡された引数は@_という特殊な配列にセットされている。それをローカルのカレント参照行（$_）として定義しているのが，local($_) = @_;にあたる。

1 while というのは，置き換えが成功するかぎり繰り返し実行する常套句で，この正規表現にヒットするかぎり置換が行なわれる。

サブルーチンの返値は最後の式を評価した値になり，この場合は明示的に $_ を返している。

あるいは，これ以外にも，入力行に複数の処理を一度に行いたいときなども有効だ。たとえば1桁の数字だけを全角に置換したいときなどは，こうすればよい。

```
while(<>) {
   tr/0-9/０-９/;
   s/[０-９]{2,}/➡
      &han($&)/eg;
   print;
}
sub han {
   local($_) = @_;
   tr/０-９/0-9/;
   $_;
}
```

ここで注目したいのは，s演算子のオプションに「e」がついていることだ。このようにeオプションは，置換された文字列を式として評価することで，サブルーチンを呼び出しているんである。

サブルーチンを使ったいくつかのスクリプトをCD-ROMに添付した。第1回目の連載で紹介した「マルチファイル検索」のスクリプトでは，直接的にテキストファイルしか受け付けなかったが，フォルダをドラッグ＆ドロップすることで，フォルダ内を再帰的に（そのフォルダのなかに含まれるすべての階層のテキストファイルを）検索させるようにした。

いかにもMacPerlっぽいgrepになったと思う。

もうひとつは用語統一などのために以前作ったもので，一括ですべてを置換してしまうのではなく，（文脈的に）人間が判断すべき部分を表示して，どう置換すべきかの指示が与えられるというサブルーチンである。

実際の運用には，編集部なり著者などの好みに対応したテーブルを作る必要があるのだが，人間の査読と違って漏れがないので，便利ではないだろうか。

●サブルーチンのライブラリ化

便利なサブルーチンを作成すると，それをいろいろなスクリプトから使いたくなるのが人情だ。そして，（当然ながら）サブルーチンをいちいちコピー＆ペーストするのがおっくうだというリクエストにも，Perlは応えてくれる。とことんラクしたいんである。

CD-ROMに添付されている「MTSH.pl」をMacJPerlアプリケーションと同じ階層にある「lib」フォルダに入れたら，以下のスクリプトを実行してみてほしい。

```
require "MTSH.pl";
$cen = "1999";
$wa=&MTSH'conv($cen);
print "$wa¥n";
```

「平成11」と返って来ただろうか？　そう，これは，西暦を渡すと和暦を返すサブルーチンをライブラリ化してしまったのだ。

重要なのは1行目，特定のライブラリを使用する場合は「require "〈ライブラリ名〉";」の1行を最初に入れておかなければならない。これでこのライブラリは，このスクリプト中に書かれたも同然に取り込まれ，どこからでも呼び出すことが可能になる。

すでにデフォルトで，たくさんの便利なライブラリがlibフォルダ内にバンドルされているので，これらを使いこなすだけでも，たいへん便利な機能をカンタンに実現できるようになるんである。

MacPerl 入門

まかせて!! MacPerl

編集者・デザイナー・DTPオペレーターのためのMacPerl (5)

CD-ROM

市川せうぞー●株式会社シンクス

**あなたが日常的にやっている多くの
アナログな仕事のいくつかを
カンタンに，素早く，オートマティックで，
便利にこなしたいと考えているなら
ズバリPerlをオススメしよう
Perlはあなたが思っている以上に
柔軟にあなたの力になるにちがいない
そんなPerlを紹介するページ
*"You Got Power!!"***

今月のテーマ

MacPerl拡張

MacPerlにはMacならではの拡張が施されている。MacPerlを使うなら，すこぶる便利なこんなの機能を使わないのはもったいないじゃない？

Macは洗練されたGUIを持つ。そんなこといまさら言われなくたってMacユーザーなら誰でも知っていることだ。そして，MacPerlもしかりなんである。

もともとUNIXで生まれたPerlは,Macに移植される際，Macの持つすぐれたユーザーインターフェイスをうまく利用できるよう工夫・拡張された。Perlのコアも当然すばらしいものなのだが，MacPerlは，それをMacで使用するにふさわしい形で提供することに成功している，まったくMacらしいアプリケーションといえるのだ。

ドラッグ&ドロップで処理ファイル（@ARGV）を受け渡すドロップレット形式や，Perlアプリケーションがなくても動作するランタイム形式も，いかにもMacらしい拡張だが，スクリプト上で使われる，便利でエレガントなテクニックのいくつかを紹介することにしよう。

メッセージダイアログ

まずは簡単なメッセージを表示してみる。

```
$mess = &MacPerl'Answer(
"あなたの性別は？",
"秘密","男","女");
print $mess;
```

図1のようなダイアログが表示されたのを確認してほしい。このようにボタンは3つまで設定でき，ソースコードで右から0～2までを返す。この場合，「秘密」ボタンを押せば「2」が返り，「男」ボタンを押せば「1」が返り，「女」ボタンを押せば「0」が返る。このようなユーザーからの入力によって動作を分岐させられるようなインタラクティブな設計が可能になる。

このメッセージダイアログの応用として，ダブルクリックするとスクリプトの説明をするようにしてみよう。

```
unless (@ARGV) {#ダブルクリックされた
&MacPerl'Answer(<<'mess');
このスクリプトは毒にも薬にもなりません
ファイルをドラッグ&ドロップしてください
(c)1999 いちかわせうぞー
mess
exit;#プログラムの終了
}
```

これは，@ARGVが空のとき「mess」という任意の文字列が現われるまでをメッセージダイアログとして表示している（図2）。この場合，処理ファイルがないことを想定しているので，exit関数でスクリプトを強制終了させている。

図1●ボタンダイアログ

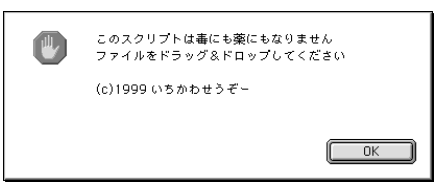


図2●メッセージダイアログ

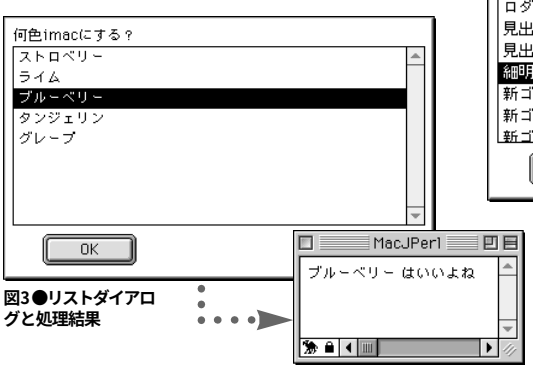


図3●リストダイアログと処理結果

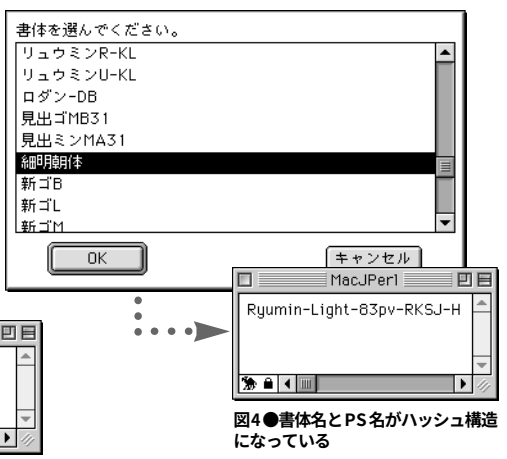


図4●書体名とPS名がハッシュ構造になっている

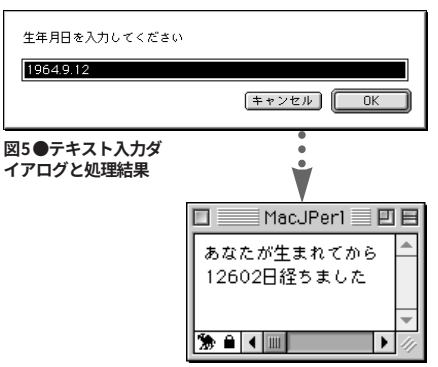


図5●テキスト入力ダイアログと処理結果

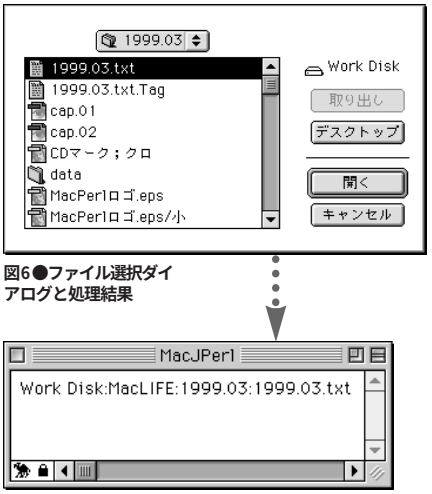


図6●ファイル選択ダイアログと処理結果

図7●フォルダ選択ダイアログと処理結果

図8●ファイル保存ダイアログと処理結果

リストダイアログ

たくさんの選択肢からひとつをユーザーに選ばせたいなら，このリストダイアログが大変便利だろう。

```
@variation = ("ストロベリー",
"ライム","ブルーベリー", "タンジェリン",
"グレープ");
$color = &MacPerl'Pick(
"何色imacにする?", @variation);
print "$color はいいよね";
```

図3のように，選ばれた文字列を返す。この応用例として，MACLIFE CD-ROMに添付した「04_Pick.2」を参照していただきたい。このスクリプトはフォント名を選択すると，フォントのPS名を返す(図4)。このように，ハッシュのキーを選択させて，その値を取り出すことも可能だ。

テキスト入力ダイアログ

ユーザーから入力されたテキストをパラメータとして取り込むという手法も，スクリプトを汎用化させるのにとても便利な方法だろう。

```
require "timelocal.pl";
($year, $mon, $mday) = split(/¥./,
&MacPerl'Ask(
   "生年月日を入力してください",
   "1964.9.12"));
$time = int((time - &timelocal
   (0,0,0,$mday, ($mon - 1),
   $year))/86400);
print "あなたが生まれてから¥n➡
   $time日経ちました¥n";
```

注）スクリプト内の➡印は，改行せずに下の行が続くというマークです。

図5のように生年月日を入力すると，生まれた日から何日たっているかを計算してくれる。しかし「今日まで一体なにをしてきたのだろう？」と頭をかかえてしまわないように。

ファイル(フォルダ)選択ダイアログ

スクリプトからのファイルの選択やフォルダの選択，新規保存ファイルのパスの生成は目立たない機能だが，独立したアプリケーションを作成する際には必須の機能だといえる。

ドラッグ＆ドロップではなく，スクリプト中で明示的にファイルを選択したい時，ファイルを選択する，こんなダイアログを使えばよい。

```
require "StandardFile.pl";
@types = ("TEXT","EPSF","TIFF");
$filename = ➡
   &StandardFile'GetFile(@types);
print "$filename¥n";
```

これで図6のようなダイアログが開く。@typesで表示するファイルのファイルタイプを制限できるのだが，引数を省略すれば，すべてのファイルを表示するようにできる。

特定のフォルダ内を処理対象にしたい時や，処理後のファイルの格納場所としてフォルダを指定したい時，フォルダを選択できるダイアログを使う。

```
require "StandardFile.pl";
$foldername = ➡
   &StandardFile'GetFolder
   ("フォルダを選択してください");
print "$foldername¥n";
```

図7のように，フォルダを選択するダイアログが開く。

処理結果をファイルに書き戻すとき，新規ファイルを保存するダイアログが開いて，任意の場所に任意の名前で保存することもできる（図8)。

```
require "StandardFile.pl";
$newname = ➡
   &StandardFile'GetNewFile
   ("どこに保存しますか?", "new.txt");
print "$newname¥n";
```

しかし，ここでファイルが生成されたわけではない。この時点では作られるべきファイルのフルパス名が渡されただけで，実際には，ファイルハンドルを開いてファイルに書き込みが行なわれなければならない。そこで初めてファイルが生成されるということを覚えておこう。

ファイルタイプとクリエータ

Macでは書類を識別するために，そのファイルの性質を表わすファイルタイプと，その書類を作ったアプリケーションが持つ固有のファイルクリエータが設定されている。YooEditの書類なら，ファイルタイプが「TEXT」，ファイルクリエータが「YoED」という英数字4文字ずつで構成されている。

これらを自由に取得したり設定したりすることで，アプリケーション固有の書類を（ファインダの見かけ上）作ることができる。

たとえば，ドラッグ＆ドロップされたファイルをすべてYooEdit書類にするためには，

```
&MacPerl'SetFileInfo( ➡
   "YoED", "TEXT", @ARGV);
```

MacPerl 入門

まかせて!! MacPerl

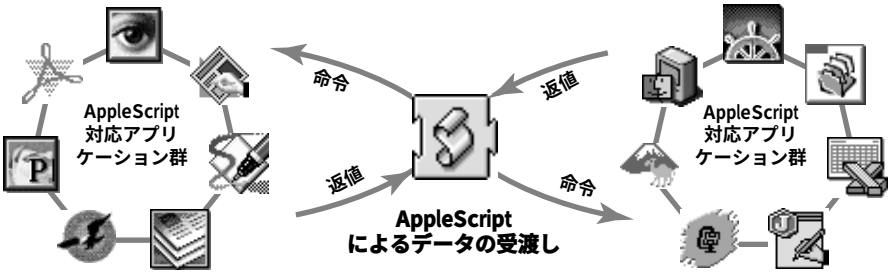


図9●PerlスクリプトからAppleScriptダイアログを呼び出す

たった1行で，便利なユーティリティとして機能してしまう。恐るべし，MacPerl。

この応用として，MACLIFE CD-ROMに添付した「10_ドロップレットの雛形」を参照していただきたい。日常的な処理のなかで書き出すファイルのタイプとクリエータが，ドラッグ&ドロップされたファイルと同じものになる仕様であれば，より便利で自然なふるまいとなる。

AppleScriptとの連携

異論はあると思うが，OpenDocは実に実によくできたテクノロジーだったと思う。最近注目を浴び始めたAppleScriptも，実装当初はOpenDocのコンテナパート間の通信をさせる仕組み（Apple Event）の布石にほかならなかった。

AppleScript対応アプリケーションとは，単にひとつのアプリケーションを外部から（スクリプト的に）操作できるだけではない。対応アプリケーションどうしはAppleScriptによってデータの受け渡しをして，互いに不足する機能を補完できる。誤解を恐れずに言うなら，「完全万能型の大きなアプリケーションから小さなアプリケーションによる協調作業」というOpenDocの思想の一端は，脈々と息づいているんである。

そしてMacPerlも，いち早くAppleScriptに対応したアプリケーションなのだ。たとえば，Jedit2やQuarkXPressなどにはソートエンジンは実装されていない。もちろん標準ではAppleScript自身にもソート関数はない。しかし，Jedit2書類の選択範囲をAppleScript経由でMacPerlに渡してソート済みのデータに置き換えることは可能なのだ。実際にCD-ROMに「Jedit sort」を添付したので，この便利さを実感してほしい。

PerlScriptにAppleScriptを埋め込む

MacPerlは単なるAppleScript対応アプリケーションではない。Perlスクリプト中にAppleScriptをそのままの形で挿入することができるんである。なんだか無性にうれしい（図9）。

```
&MacPerl'DoAppleScript(➡
  <<END_SCRIPT);
tell application "MacJPerl"
  set ans to (display dialog ➡
  "入力待ち" default answer ➡
  "print 'Howdy world';" ➡
  buttons {"キャンセル", "書込のみ",➡
  "実行"} default button 3 ➡
  with icon 1)
  make new Window
  copy (text returned of ans) to ➡
  character 1 of front Window
  if (button returned of ans) = ➡
  "実行" then
    Do Script (text returned of ans)
  end if
end tell
END_SCRIPT
```

どこからがAppleScriptワールドで，どこまでがPerlワールドなんだか混乱してしまうかもしれない。というかこれではほとんどAppleScriptそのものだ。

&MacPerl'DoAppleScriptから指定された引数「END_SCRIPT」が出現するまでがAppleScriptとして動作する仕組みになっている（引数は任意の文字列）。このコマンドに挟まれた部分は完全にAppleScriptそのままだ。Perlスクリプト中にAppleScriptを記述できる利点は，操作したい対応アプリケーションとの間にAppleScriptのアップレットを介さなくてよいということにほかならない。ちょうど，ファイルメーカーにAppleScriptを埋め込むのと似ている。

AppleScriptにPerlScriptを埋め込む

では逆にAppleScriptにPerlスクリプトを埋め込むことはできるのだろうか？　できるのだ（図10）。

```
set data_t to "123円の買い物をして➡
  500円払ったら？"
set Perl_script to "
@data = split(/[^¥¥d]+/, $ARGV[0]);
$c = ($data[1] - $data[0]) . '円です';
&MacPerl'Reply($c);"
tell application "MacJPerl"
  set kekka to Do Script ➡
  {Perl_script, data_t}
  display dialog data_t & ➡
  return & kekka
end tell
```

AppleScript中にPerlスクリプトを記述する時，気をつけなければならないことがある。AppleScriptにとって「¥」と「"」は特別な意味を持つキャラクタなので，「¥」を使ってエスケープしてやることを忘れてはいけない。

データの受け渡し

そろそろタネ明かしの時間だ。これらのマジックの秘密は2つある。ひとつは，AppleScript側からPerlに命令を送ることのできる「Do Scriptコマンド」。もうひとつは，Perlスクリプト側からAppleScript側へ値を渡す「&MacPerl'Reply(値);」である。

Do Scriptコマンドは，大きく分けて2つの引数をとる。1番目の引数がスクリプトそのものを指定しており，上記の例ではAppleScript中にPerlスクリプトをそのままAppleScriptの変数に

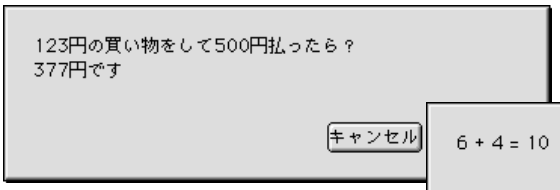


図10●MacPerlから計算結果を受け取る

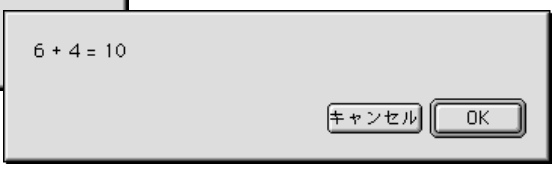


図11●mode Batchオプションを使った例

筆者紹介

市川せうぞー

1964年9月12日生まれ，B型。
筋金入りの花粉症。外出時には完全武装を欠かせない。JAGAT(日本印刷技術協会)認定・DTPエキスパート。
http://www.asahi-net.or.jp/~ym3s-ickw/showtime.html

●ご注意●

今月号よりCD-ROMに添付のサンプルスクリプトはすべてMacJPerl5.2.0r4 J1に準拠しました。MacJPerl5.2.0r4 J1はCD-ROMに収録されております。

MacJPerl5.2.0r4 J1からクリエータ等が変更となったため，MacJPerl5.2.0r4 J1で作成されたドロップレット形式・ランタイム形式・CGI形式はMacJPerl5.2.0r4 J1がなければ開くことはできません（テキスト形式は大丈夫）。また，単にクリエータを「MrPL」などしても旧バージョンおよび英語版バージョンではソースを見ることはできません。

どうしてもMacJPerl5.2.0r4 J1以外で作業したいかたもおられると思います。そういう場合はResEdit等のリソースエディタでTEXTリソースの#128をコピー＆ペーストしていただければいいかと思います。

逆に，旧来のバージョンで作成されたものはMacJPerl5.2.0r4 J1でそのまま編集できます。お手数をおかけしますが，なにとぞご了承ください。

入れて，Do Scriptコマンドで Perlスクリプトを直接MacPerlに送っているが，この引数はPerlスクリプトのフルパス名を渡せば，そのスクリプトを起動させてくれる。

Do Scriptコマンドの2番目以降の引数（リスト）はすべてPerl側では@ARGVに格納されており，入力データとして利用できる。Perl側で$ARGV[1]とすれば，Do Scriptコマンドの3番目のリストの引数を参照する。

さらにDo Scriptコマンドのenvironmentオプションを利用して，AppleScriptのリストをPerlにユーザー定義の環境変数として渡すこともできる。詳しくは，MACLIFE CD-ROMに添付した「15_environment」を参照していただきたい。

Perl側での処理結果をAppleScriptに返す方法としては「&MacPerl'Reply(値);」というコマンドが使用できる。この場合，返すことのできる値はスカラーだけなので，配列として受け取りたい場合は「AppleScript's text item delimiters」などを使用してリストに戻すテクニックが必要になる（右下のスクリプトを参照）。

「&MacPerl'Reply(値);」以外にも，「Do Scriptコマンド」に「mode Batch」オプションをつけることで，データを通常のSTDOUTへのprintでそのままAppleScriptに送ることもできる。

またPerl側でeval関数を使って，渡された文字列（データ）をPerl式として評価することも可能だ（図11）。

```
set str to "6 + 4"
set perl_script to "
$ans = eval($ARGV[0]);
print $ans;"
tell application "MacJPerl"
   set ANS to Do Script ➡
   {perl_script, str} mode Batch
   display dialog str & " = " & ANS
end tell
```

先月もご紹介したように，eval関数は文字列をPerl式として評価する。この関数がPerlにあるおかげで，AppleScriptとの連携は柔軟性が格段に向上するのだ。

MacPerlが単なるAppleScript対応アプリケーションではないことがおわかりいただけたと思う。MacPerlはAppleScriptと連携することによってMac環境での最強のテキスト処理を実現してくれる。これらの方法を駆使すれば，その可能性は想像の数だけ膨らむにちがいない。

もちろん，これらの機能はMacPerl独自のものであり，他のUNIXやWindows環境では動作しない。そういった意味でPerlの純度は損なわれるかもしれないが，この便利さにはそれらの欠点を補ってあまりあるものがある。Macはコマンドライン・インターフェイスではないのだ。そう，Macにはエゴイスティックなまでにそう断言させてしまうなにかがあるのかもしれない。そしてMacPerlにも。

今月紹介させていただいたMacPerl拡張に関する便利なテクニックは，MacPerlアプリケーションと同じ階層にある「MacPerl.Specifics」に詳しく記述されている。今回それを翻訳して「MacPerl.Specifics_J」というファイルをCD-ROMに添付した。誌上での紹介に漏れた機能もいくつかあるので参照していただきたい。

```
####AppleScript側
set aList to {"３５９１１３１", "Ｉｃｈｉｋａｗａ", "せうぞー", "（有）しんくす"}
set F_path to choose file with prompt "test.plを選択してください"
tell application "MacJPerl"
   set aRetList to Do Script ({F_path} & aList)
end tell
set AppleScript's text item delimiters to {"★★★"}
set aList to (every text item of aRetList) as list
set AppleScript's text item delimiters to {""}
aList
```

```
####PerlScript側（test.pl）
#! /usr/local/bin/perl
foreach $_ (@ARGV){
   tr/ぁ-んＡ-Ｚａ-ｚ０-９有ァ-ンA-Za-z0-9株/;
}
$ans = join('★★★',@ARGV);
MacPerl::Reply($ans);
```

```
####AppleScript結果
{"3591131", "Ichikawa", "セウゾー", "（株）シンクス"}
```

MacPerl 入門

まかせて!! MacPerl

編集者・デザイナー・DTPオペレーターのためのMacPerl (6)

CD-ROM

市川せうぞー●株式会社シンクス

あなたが日常的にやっている
多くのアナログな仕事のいくつかを
カンタンに，素早く，オートマティックで，
便利にこなしたいと考えているなら
ズバリPerlをオススメしよう
Perlはあなたが思っている以上に
柔軟にあなたの力になるにちがいない
そんなPerlを紹介するページ
"You Got Power!!"

今月のテーマ

ケーススタディ

さまざまな場面で活躍しているPerlプログラムを紹介しつつ，その原理を簡単に解説する。連載最終回。

有史以前のハッカーの先祖は言った。「他人のプログラムをそのまま盗むことを"パクリ"といい，解析しうまく取り込むことを"学習"という」と。

多くの有益なソースコードを読むことは，必須科目の修行であり，プログラム上達の最短の近道なんである。

幸い，何人かの優秀なプログラマーの協力を得ることができ，いくつかの有益なサンプルを提供していただいた（MACLIFE CD-ROMに添付）。改めて感謝の意を表したい。

今回はその中のいくつかを紙面で紹介してみようと思う。これらの有益なプログラムは使用するだけでも便利なものばかりなので，ぜひご活用されたい。

QuarkXPressとの連携

レイアウトソフトで文字組みをするとき，スタイル属性や文字属性，縦組みにおける文字種の選択など，膨大で退屈で面倒な作業から逃れることはできない。

その解決策として，主なレイアウトソフトでは，独自のマークアップタグが用意されており，文字組属性のほとんどを再現できるようになっている。

ただし，このタグはお世辞にも人間が「見やすく扱いやすい」という観点は欠落している部分があり，どうやってマークアップするのか？という問いにマニュアルは答えていない。もしかして，人が入力するのだろうか？

そこでPerlの登場なんである。Perlの正規表現置換はいともカンタンにこれらの問題を解決してくれる。実際に筆者が関わる書籍の仕事のほとんどはPerlのお世話になっている。

たとえば章・節・項見出しにそれぞれのナンバーが振られているとして，XPress Tagを使ってスタイルを適用させるならこんな感じ。

```
#!/usr/local/bin/perl -pi.bak
s/¥¥/<¥¥¥¥>/g;
s/<([^¥¥])/<¥¥<>$1/g;
s/@/<¥¥@>/g;
s/^¥d¥t/@章見出し:<¥$>$&/;
s/^¥d¥.¥d¥t/@節見出し:<¥$>$&/;
s/^¥d¥.¥d¥.¥d¥t/@項見出し:<¥$>$&/;
```

DTP専門誌の記事でさえ，これをエディタのマクロで組みましょうなどと推奨しているが，なにが悲しくてそんな回り道をする必要があるのだろう。Perlなら30秒で書いて1秒で処理が終わるというのに。

まず，最初の3行はXPress Tagならではのおきまりで，タグに使われるメタ文字をエスケープしている。あとは，「s//;」演算子でひたすら特定のパターンにタグを付加していくだけだ。

QuarkXPress & MacPerl Scripts

いったんこういう便利なやり方を知ってしまうと，人間はさらにラクをしたくなるんである。テキスト段階で前処理加工をするのではなく，QuarkXPressのドキュメント上でテキストを自由に扱いたいという欲求が生まれてくる。そして，それができると信じるようになる。

やがてラクをできるという信念を実現する人々が出てきた。NIFTYSERVEのDTPフォーラム（FDTP）15番会議室（せど・おうく・ぱある）で発表され（1343番からのスレッド），おもに青山和光氏と鎌田幸雄氏によって改良が加えられ，現在もフリーウェアとして公開されている。

このスクリプトを簡単に説明すると，およそ図1のようになっている。まずAppleScriptを使ってドキュメント上で選択されたテキスト（あるいはカレントのテキストボックスのストーリー）を受け取ってPerlモジュールに渡す。Perlモジュールは受け取ったデータを処理しファイルに書き出す。AppleScriptはそれを受け取って元のドキュメント上に挿入するという手順を踏んでいる。つまり，このスクリプトが提供しているのはドキュメント上のデータをPerlモジュールに受け渡す仕組みであり，モジュールさえ追加すれば機能はどんどん増えていくのだ。

ドキュメント上のタグを受渡しする仕組みも「xtag.pl」がライブラリ化され配慮がなされている。したがってスクリプトそのものは，いたってシンプルに記述することができる。

具体的な操作方法の流れを図2で示した。特別な知識はほとんど不要なのがおわかりだろう。

今回CD-ROM MACLIFEに収録したモジュールは以下のとおり。

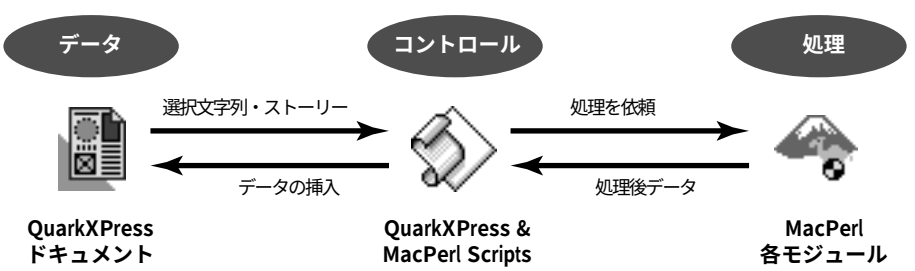


図1●QuarkXPress & MacPerl Scriptsのおおまかな流れ

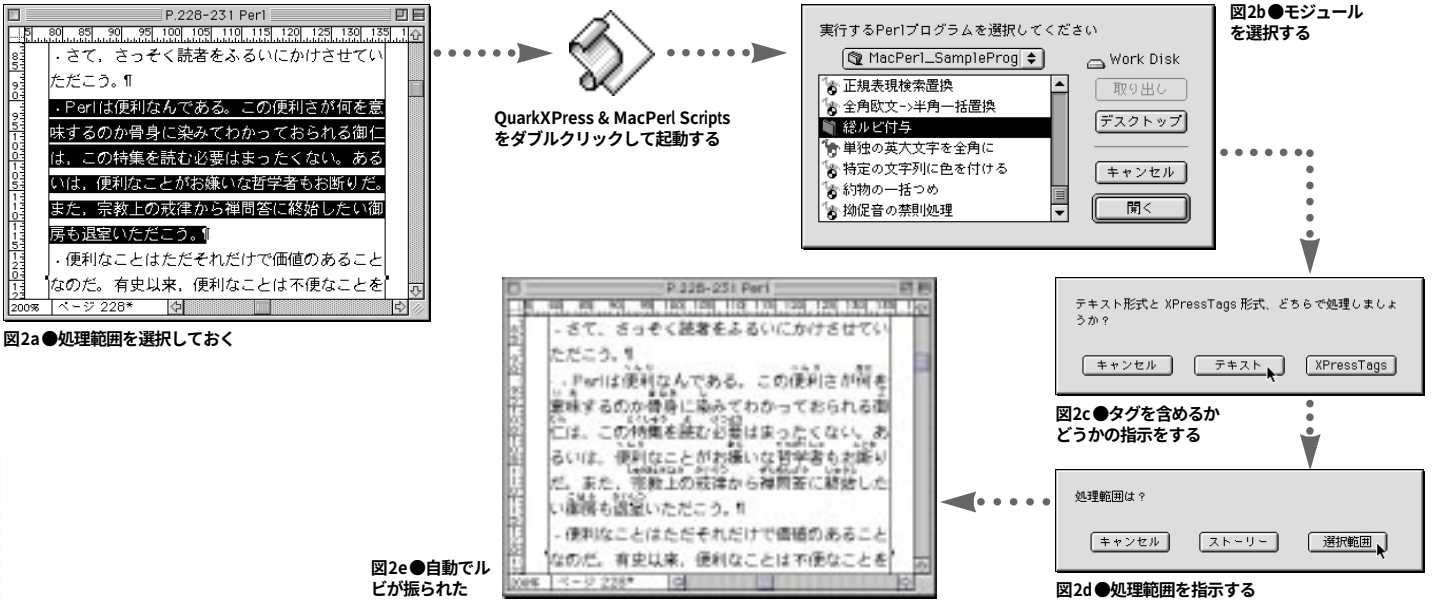


図2a●処理範囲を選択しておく

図2b●モジュールを選択する

図2c●タグを含めるかどうかの指示をする

図2d●処理範囲を指示する

図2e●自動でルビが振られた

・ビブロスフォントの一括指定
・英数字の全角半角統一
・行ソート
・行ナンバリング
・行頭約物前に半角アキ
・縦組中の半角数字回転(桁無制限)
・縦中横文字
・正規表現検索置換
・全角欧文->半角一括置換
・総ルビ付与
・単独の英大文字を全角に
・特定の文字列に色を付ける
・約物の一括つめ
・拗促音の禁則処理

どのモジュールもQuarkXPressの標準では実現しない，かゆいところに手が届くようなものばかりである

そしてこのスクリプトがすばらしいのは誰でもがスクリプトに手を加えたり，新たなモジュールを簡単に追加できることだ。XTentionが便利だといっても，こういうことはできないだろう。

HTMLもラクラク一括処理

「マークアップ」や「タグ」という言葉でHTMLを連想される方も多いと思う。

DTPと一言でいっても「紙」だけが舞台ではない。「電子出版」という言葉が日常語となっているように，現在ではインターネットやWeb空間を無視することはできない。

当然の欲求として印刷物としてのデータをそのままWeb上に生かす，いわゆるワンソース・マルチユースという考え方も，現在では実践段階を迎えている。

そしてここでもPerlが大活躍をする。大量のドキュメントを同じ体裁でHTML化するとしても，もう頭を抱える必要はない。

原理はXPress Tagと同じで，ある特定のパターンにタグを付加していくだけだ。すでに1月号のCD-ROMに収録したとおり。

タグに使用されるメタ文字としてこれらをエスケープしなければならないものXPress Tagと同じである。

```
s/¥&/¥&amp;/g;
s/</¥&lt;/g;
s/>/¥&gt;/g;
s/¥"/¥&quot;/g
```

プレーンテキストからHTMLへのコンバータとして，中島 靖氏作の「txt2html.pl」を汎用的な例解としてCD-ROM MACLIFEに収録した。このスクリプトをカスタマイズすれば，かなりの作業が自動で行なえるはずだ。このスクリプトを使うと，HTMLエディタを使うことさえおっくうになるかもしれない。

この逆に，もしかしたら，あなたのクライアントは大量のHTMLファイルを持ち込んで，「全部プレーンテキストに直してほしい」と言いだすかもしれない。それがブラウザの「テキストで保存」ではとても追いつかない量だとしても，あなたは笑顔でその仕事を受けることができる。

そのHTMLが単純な構造なら，

```
#!/usr/local/bin/perl -pi.bak
s/<[^>]+>//g;
s/^¥s*¥n//;
s/¥&amp;/¥&/g;
s/¥&lt;/</g;
s/¥&gt;/>/g;
s/¥&quot;/¥"/g
```

などとすればよい。しかしHTMLはタグの中でも改行が許されていたり，文中での改行も割といい加減だったりする。

もう少し高度なコンバータとして，同じく中島靖氏作の「html2txt.pl」を収録した。ブラウザが吐き出すテキストよりも整然と整形されたテキストを書き出してくれる。

IllustratorやPSだってテキストだったら大丈夫

DTPの最も基盤となる技術はPostScript（以下PSと略）である。DTPの黎明期にはこのPSに精通し，プログラム的に作表したり，組み版したりできる無形文化財的ツワモノがいたらしい。

PS自体は汎用言語という側面もあり，データとしてPSを扱う場合，少なからずPSの知識が必要なのだが，AdobeはPSの技術資料を公開しているため，解析できないというわけではない。むしろ，言語としての構文や約束事が守られているぶん扱いやすいともいえる。

たとえばPS上で各ページは“%%PageBegin〜%%PageEnd”の中に記述されていて，それぞれ相対ページと絶対ページを持つため，それを利用して，PSから索引を作ることもできる（CD-ROM収録「PS_Index.pl」）。PSから索引を引っ張る利点は，レイアウトソフト上に貼り込まれたドローデータ（IllustratorやFreeHandなど）からも，語彙を収集できることだ。

他にもPS中にはフォント情報やサイズ情報など出力に必要な情報が整然と詰め込まれているため，条件を限定してテキストを抽出することも可能だ。たとえば，「Picup Text of Illustrator」（CD-ROMに収録）ではIllustrator形式のファイルから「10ポイントの中ゴシック」のテキストだけを取り出すこともできる（図3）。

ここでちょっと厄介な問題が生じることがあるかもしれない。PSは伝送路やプロトコルの関

MacPerl 入門

まかせて!! MacPerl

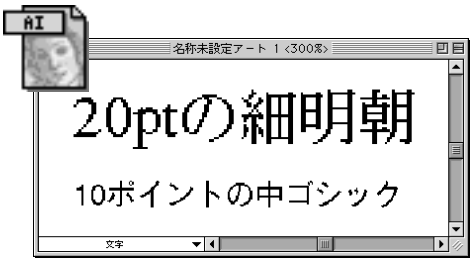


図3●Illustratorのファイル中の10ポイントの中ゴシックのテキストだけが抽出された

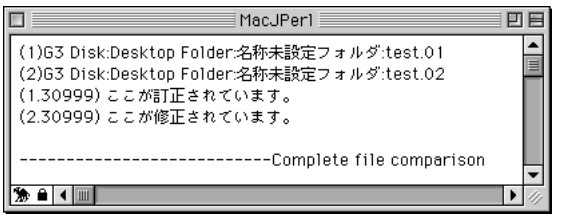
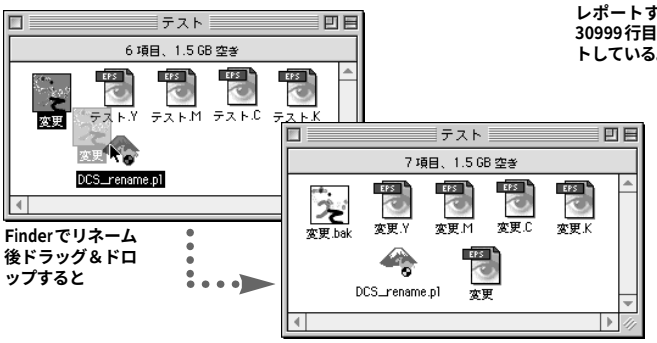


図4●ふたつのファイルをドラッグ&ドロップするだけでその違いをレポートする。このサンプルでは30999行目の1文字の違いをレポートしている。処理時間は約2秒



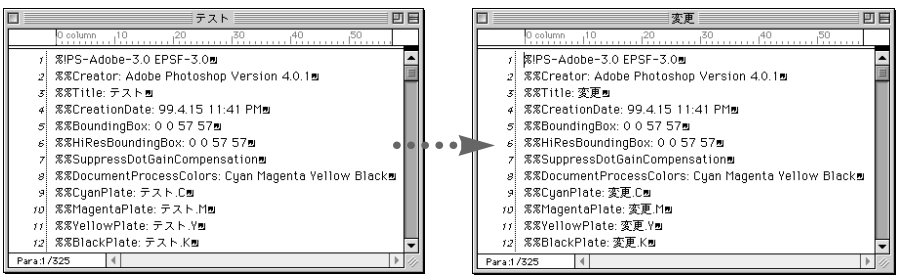


図5●ファイルの中身も書き換えられているのがわかる

係で2バイトの文字を8進（あるいは16進）数で表現することが多い。

しかしPerlは8進（あるいは16進）数を直接パックする関数を持っているため，下記のようなサブルーチンにを作ってやれば，8進数で表現された文字列オブジェクトを簡単にデコードすることができる。

```
sub PP {
local($_) = @_;
s/¥¥¥¥/¥¥134/g;
s/¥¥(¥d{3})¥¥(¥d{3})/
   pack("CC",oct($1),oct($2))/eg;
s/¥¥(¥d{3})/pack("C",oct($1))/eg;
s/¥¥r/¥n/g;
$_;
}
```

これらの「pack」や「unpack」のサンプルとして「word conv」をサンプルとして添付したので参照していただきたい。これらの文字列コンバートのサンプルはPSファイルの中身を操作する上でもきっと役立つだろう。

さらに複雑な文字コードの処理が必要な場面では歌代和正氏作の文字列変換ライブラリ「jcode.pl」（CD-ROM添付）が活躍するだろう。「jcode.pl」はもともと英語版Perlで日本語を扱うために開発されたライブラリで，日本語処理環境では標準ライブラリといってもよいかもしれない。

ほかにもIllustrator形式はASCIIデータなので，Perlが役立つ場面が多い。青山和光氏作の「イラストレーター5.0Jの文字のせ」「Illustrator5.0Jでルビ文字作成」「文字だけを別レイヤーに集める」（CD-ROMに収録）などはその魅力の一端を知ることができる。

便利なスクリプトの数々

●テキストファイルの差分をレポートする

データ管理をする上で，時として「修正されているかもしれない」というテキストが発生することがある（あまり好ましいこととはいえないが…）。オリジナルとよく似ているがタイムスタンプが違ったり，グループワークの途中で新たなデータが書き加えられたりするような場面は日常茶飯事的に起こりうることではないだろうか。

さてそんなとき，あなたならどうする？　ふたつのドキュメントを開いて査読するしかないのだろうか？　そんなこと，人間のすることじゃないと思うあなたは正しい。そしてPerlの登場なんである。

侭田和広氏作の「textdiff.perl」（CD-ROMに収録）はふたつのテキストファイルの違っている部分をレポートしてくれる（図4）。

●DCS 5ファイルのリネーム

DCS 5ファイル形式は，プリントサーバにカラーの画像データを転送する際に，ネットワークへの負荷を軽減する有効な手段である。また，RIPでの処理速度向上にも貢献する。

しかし分割されたファイル名の関係がPS内部にも記述されているため，リネームすることは許されていない。もし誤ってリネームしてしまった場合……その結果はみなさんもご存じのとおりだ。

青山和光氏作の「DCS_rename.pl」（CD-ROMに収録）を使用すればこれらのファイル名の関係を正しく修正してくれる（図5）。もちろん，名前を変更しないことが原則なので，あえてリスクを負うこともないのではあるが，こういう原則が行き渡らない部署が存在するということもまた，事実だったりする。

●ファイルの分割と連結

ファイルの分割や連結は日常的に行なわれる

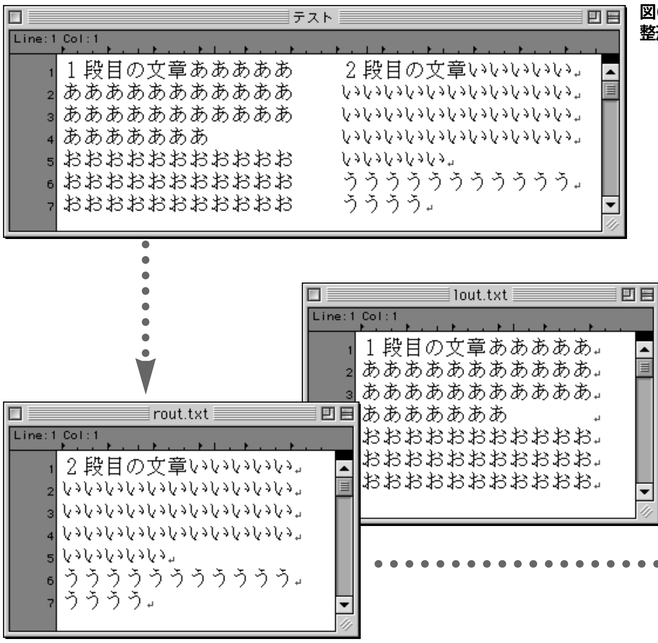


図6●テキストの整形も自在

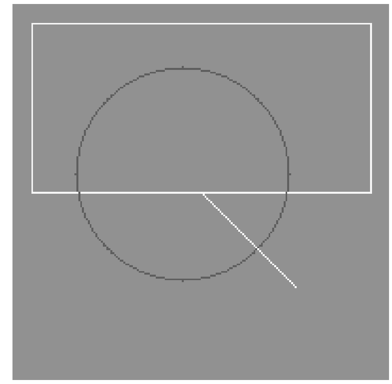


図7●Perlが描画して出力したGIF画像。カラーでお見せできないのが残念だ

業務だといえる。原稿の整理やメールの管理などでも活躍するだろう。こういう整理ができているかいないかで，その後の仕事のしやすさが変わってくるというものだ。そしてそんな簡単なことは簡単にPerlでやっつけてしまおう。

「ファイル連結」（CD-ROM添付）は単にファイルを連結するだけでなく，該当ファイルの先頭行にファイルの先頭を表すデリミタ（任意の文字列）とフルパス名を書き込んでくれる。あとでそのデリミタをサーチすれば，ひとめでファイルの区切りが分かるようになっている。

「ファイル分割」（CD-ROMに収録）ではそのデリミタを手がかりに自動でファイルを切り分けてくれる。デリミタの直後にフルパス名かファイル名が記述されていれば，その名前を使用する(つまり「ファイル連結」と互換している)。

「空行でファイル分割」（CD-ROMに収録）では，デリミタでの分割ではなく，空行をサーチしてファイルを分割してくれる。

これらのスクリプトは飯田重規氏からヒントをいただいた。

●整形されたテキスト

ワープロなどできれいに整形されたテキストにお目にかかることがある。ワープロ上で2段組にされていたり，文字詰めを調整されていたりする「アレ」である。たぶんこれを作った人は「こうするときっと手間が省けているんじゃないか」と思っているかもしれない。これは大きな誤解で，大抵の場合，こういうテキストは清書され直されているんである。

これも人間がやっていて決して面白い作業じゃない。しかしPerlを使えば，問題はたちどころに解決する。青山和光氏作の「２段組みワープロデータの前処理」「行末改行を取る」（CD-ROMに収録）など参照していただきたい（図6）。

●SMTPサーバにメールを送る

SMTPとメールの送受信についての詳しい解説は本誌連載中の「実験自作のメーラアプレット」の田中先生にお譲りするとして，とにかくPerlでも，簡単にネットワークのソケットへデータを投げることができる。

山本 武氏作の「MacPerlから電子メール」（CD-ROM添付）を参照していただいただければ分かるとおり，メールとして最低限の情報が揃っていれば，メールは簡単に送ることができることが実感できる。実際に「sample_mail」をドラッグ＆ドロップすれば，あっという間に送信が完了しててしまう。通常のメーラとなんら違いのない手軽さだ。

●GIFを描画する

Perlを使ってGIFを描画するというのは，なんだか新鮮な驚きを隠せない。

諏訪園秀吾氏作の「GIFを描画」（CD-ROMに収録）を実行してみてほしい。図7のようなGIFが出力される。

たとえば，CGIプログラムに組み込んで，Web上で刻々と変化する棒グラフなどを見せるということも可能になるだろう。

“便利”ってなんだ?!

「便利」という切り口でMacPerlを6ヶ月間紹介してきた本連載も，最終回を迎えることとなった。MacPerlの魅力のすべてを伝えられたかどうか，はなはだこころもとない。

Perlはスクリプト言語である。「難しそう」「面倒くさそう」「プログラマだけが使うもの」という印象がどうしてもつきまとってしまう。しかし，使ってみてわかるとおり，実際はだれもが簡単に使える便利なものなんだが。

そもそも「便利」ってなんだろう？　「不便」の反語であろうか？　コンピュータは大変便利な道具である。しかし同時にたいへん不便な道具でもあることは否定できない。DTPは商業印刷の主流となったが，職人の職域を完全にデジタル化できない，脆弱なシステムから抜けきれないでいる。あらゆる「便利」なものは必ず「不便」を持ち合わせるといっていい。人間が「不便」から「便利」を創造する生き物であるならば，このサイクルから脱出する手段はない。林檎はもうかじられているのだ。

「便利」ってなんだろう？　Perlはそのひとつの解答だと思う。そしてかなり「まとも」な解答なのだ。

これまでおつきあいいただいた読者に感謝します。ありがとう。

筆者紹介

市川せうぞー

1964年9月12日生まれ，B型。
太っている。昔の友達に久しぶりに会ったりすると「太ったねー」と言われる。実は妻も太っている。ついでにいうと猫も太っている……。“しあわせ”なんである。
JAGAT(日本印刷技術協会)認定・DTPエキスパート。
http://www.asahi-net.or.jp/~ym3s-ickw/showtime.html