

計算機科学'13 #4 — スクリプトとテキスト処理

久野 靖*

2013.6.11

1 スクリプトとスクリプト言語の役割

1.1 スクリプトの発祥: シェルスクリプト

スクリプト (**script**) とはもともと「台本」のことで、コンピュータの世界では「実行しなければならないことを順番に書き連ねたもの」といった意味で使われてきました。その一番最初は、Unix のコマンドをファイルに格納したものであるシェルスクリプト (**shell script**) から来ています。ただし、それ以前から OS に対する動作指示をファイルに並べて記述するものはジョブ制御言語 (Job Control Language, JCL) として知られて来ました。ただ、プログラミング言語に近い形になったのはシェルスクリプトのあたりから、ということです。

具体例で見てみましょう。たとえば、自分がある作業ディレクトリでファイルを定期的に改訂していて、ただし直し間違いがあると困るから BACKUP というディレクトリに時々ファイルをコピーする (チェックポイントを取る) ものとしします。それには、次のようなコマンドを実行すればよいでしょう (ファイルが my.txt だとします)。

```
cp mytext.txt BACKUP/mytext.txt
```

しかし、沢山のファイルについてそのつどこのちよつと長いコマンドを打ち込むのは面倒そうです。そこで、bk という名前のファイルに次の行を入れておくものとしします。

```
cp $1 BACKUP/$1
```

さらに、「chmod a+x bk」によりそのファイルの実行モードを ON にします。そうしておけば、次のようにひとこと言うだけでコピーが行えます。

```
bk mytext.txt
```

なぜそうなるかという、シェルスクリプトはコマンドを実行しますが、そのときパラメタを渡すことができるからです。渡したパラメタは \$1、\$2、… という名前で参照できます。このように、シェル内では変数 (パラメタ) を参照するには「\$」のついた番号や名前を使います。

しかし、どうせならファイル 1 個ずつを指定するのではなく、ある程度まとめて次のように指定できると嬉しそうです。

```
bk t1.txt t2.txt t3.txt    ← 3つ指定  
bk *.txt                  ← ここにある.txt 全部
```

実はシェルスクリプトの中ではループを使うことができ、それによってこのような処理も簡単に行えます。

*経営システム科学専攻

```

for f in $*
do
  cp $f BACKUP/$f
done

```

ここで1行目の「\$*」は「パラメタ全部の並び」を表し、for文は「個々のパラメタを変数\$fに入れてdo~doneの間を繰り返し実行」という意味になります。

ところで、「*」を使って沢山のファイルをまとめてbkに渡したとき、前と変わっていないものも複数あるかも知れません。コピーが必要なものだけをコピーし、なおかつコピーするときに古いものがあつたらそれは末尾に「.old」をつけた名前にして保存すると、1つ前の版も残るのでよさそうです。そのように改良してみました。

```

for f in $*
do
  if ! test -f BACKUP/$f
  then
    cp $f BACKUP/$f
  elif ! cmp $f BACKUP/$f >/dev/null
  then
    mv BACKUP/$f BACKUP/$f.old
    cp $f BACKUP/$f
  fi
done

```

testというコマンドはさまざまな条件について調べ、その成否をシェルに返して来ます。「test -f BACKUP/\$f」では当該ファイルが有るかどうかを判定しますが、今回は「無い」場合の動作を書きたいので「if ! ...」で条件を反転しています。ifはこの条件に基づいて枝分かれし、まだバックアップが無ければバックアップを作ります。バックアップがあつた場合は、今度はcmpというコマンドで「先にコピーしたファイルと現在のファイルが同じかどうか」調べ、やはり条件を反転して「同じでない」ならば先のファイルの名前を変更した後で現在のファイルをコピーします。¹

しかしこの方法だと、常に1つ古いファイルしか取っておけませんね。もっと何回も直しても大丈夫なように、古いファイルはその変更日付がくつついて残るようにしましょう。

```

t='date +%y%m%d%H%M'
for f in $*
do
  if ! test -f BACKUP/$f || ! cmp $f BACKUP/$f >/dev/null
  then
    cp $f BACKUP/$f
    cp $f BACKUP/$f.$t
  fi
done

```

先頭の行はdateコマンドに出力形式を指定することで「年月日時分」が全部数字で出力されるようにして、その「出力を」変数tに格納します。このように、バッククオート式「`...`」を使うことでコマンドの出力を変数に取り込むことができるのです。その後は先と同様のループですが、今度は「ファイルがまだ無かったり、あつても比較して違いがあれば」常にファイル名そのままと時刻つきのファイルと両方のコピーを作るようにしていますので、プログラムは短くなりました。

¹cmpは違いのある箇所を出力しますが、今回は成否の条件だけを使うので出力は/dev/nullに送り込んで無視しています。

このような込み入ったシェルスクリプトの実行状況を把握するには、シェルを「実行表示つきで」起動してこのスクリプトを動かすようにするとよいでしょう。

```
% sh -x bk *.txt
+ date +%y%m%d%H%M
+ t=1305281713
+ test -f BACKUP/t.txt
+ cmp t.txt BACKUP/t.txt
+ test -f BACKUP/t1.txt
+ cp t1.txt BACKUP/t1.txt
+ cp t1.txt BACKUP/t1.txt.1305281713
```

これを見ると、t.txt はファイルがあったが比較したところ前と同じだったのでコピーは行わず、t1.txt はまだコピーのないファイルだったのでコピーを行った、ということが分かります。

1.2 シェルスクリプト言語のまとめ

いきなり実例だったので、シェルスクリプトのまとめを示しておきます。なお、シェルの制御構造記述は csh 系、/bin/sh 系の 2 系列に分かれていますが、ここでは後者を解説しています。²こうして見ると、コマンドを並べただけというより「言語」らしい構造を持っていることが分かります。

- 変数代入 — 変数名=値 (注意: 「=」の前後に空白を入れないこと)
- 変数参照 — \$変数名
- パラメタ参照 — \$番号 (全部まとめては「\$*」)
- コマンドの実行 — コマンド名 引数… (通常のコマンドと同じ)
- コマンド出力の代入 — 変数名='コマンド名 引数…'
- if 文/while 文/for 文 — 条件によって枝分かれする/条件が成り立つ間繰り返す/並びの値を順次変数に入れながら繰り返す

if 条件	while 条件	for 変数名 in 並び…
then	do	do
文…	文…	文…
elif 条件	done	done
then		
文…		
else		
文…		
fi		

なお、if 文では elsif は何個あってもよく、else 部は無くても構いません。

- if/while の条件には成功/失敗を返す任意のコマンドを記述でき、また条件の反転は「!」で表す。
- ファイルの有無や値の比較は test コマンドで判定できる。「test -f ファイル」(ファイルがあれば成功)、「test 値 比較演算子 値」(比較演算子が成り立てば成功) など。

²csh 系は歴史的事情もあり、スクリプトにはあまり使われません。

1.3 スクリプト言語の意義

シェルスクリプトが便利だということはお分かり頂けたかと思いますが、なぜこういうものが重要か、という点についてまとめておきましょう。

たとえば GUI である操作をやるのに、範囲を選択してマウス操作でメニューを出して選んで、それで完了、簡単でいいね、とっていたとしましょう。しかしその操作を 100 個のファイルについてやらなければならないとしたらそれは「苦行」になってしまいます。

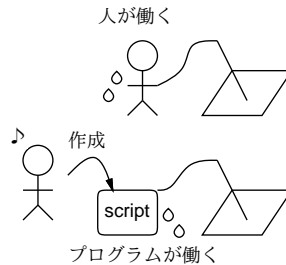


図 1: スクリプトの考え方

もちろん、そういう「苦行」から人間を解放することこそコンピュータの本来の存在意義なわけで、コンピュータのためにそんな苦行をするというのは本末転倒ですね。ではどうするか? 同じことを繰り返すのはプログラムを書いてプログラムにやらせればよいわけです (図 1)。³そしてそれを「さっさと」簡単にできる、というのがスクリプトの思想です。今日では、スクリプトを書くための言語、つまりスクリプト言語 (**scripting language**) が多く作られ使われるようになっていきます。さらに、これまでは「傍流」だったスクリプト言語を使って多くのアプリケーションが書かれるようになってきています。⁴

演習 9-1 シェルスクリプトを作る次の課題から 1 つ以上 (できれば全部) 選んでやってみなさい。

- 実行するたびに「1」「2」「3」…のように 1 つずつ大きい数値が出力されるスクリプトを `cnt` という名前で作る。最後に出力した数値を `cnt.txt` というファイルに入れておくといいでしょう。できれば、`cnt.txt` がまだ存在しない時は新たに作るとなおいでしよう。⁵
- 「`add 3`」「`add 12`」のように数値を次々に与えるとそれまでの数値の合計 (累計) が出力されるスクリプトを `add` という名前で作る。できれば、引数が 1 個も指定されていない場合は「1」不足ようになっているとなおいでしよう。累計をご破算にする方法も検討すること。⁶
- 「`pow 4 → 16`」のように、数値 N を指定すると 2 の N 乗が出力されるスクリプトを `pow` という名前で作る。できれば、「`pow -3 3 → 27`」のように、コマンドの直後に「 $-M$ 」が指定されている場合は M の N 乗になるとなおいでしよう。⁷

「◎」の条件: (1) 小問 2 つ以上やってあり、(2) いずれも「できれば」までやってあり、(3) やった内容に関する適切な説明、および適切な考察が記述されていること。

³ここでもしも、データが GUI でしか操作できないものと厄介なわけですが、Unix の文化ではデータはテキストファイルであり、スクリプトで操作できるわけです。

⁴最近ではスクリプトという単語の「簡便」というニュアンスを嫌って軽量言語 (**Light Language, LL**) という言い方もよく使われます。

⁵ヒント: 「足し算」が必要ですが、それにはコマンド `expr` コマンドを使います (例: 「`expr 3 + 1`」 → 「4」を出力)。

⁶ヒント: 「`test 値 = 値`」で値が等しいことが判定できます。引数 $\$1$ が空なら 「`x"$1" = x`」が成り立ちますね。

⁷ヒント: ループには `while` を使い、回数を入れた変数の値が 0 より大きいことを `test` で調べるのがよいでしょう。

2 Perlとテキスト処理プログラミング

2.1 文字の処理の歴史

コンピュータが最初に作られたときはその目的は文字通り「計算する」ことだったので、扱うのは数値が中心でした。文字も符合化すればビットの列で表せ、コンピュータに取り込めることは既にやった通りですが、最初のころは入出力装置がCPUと文字をやりとりするのに符合化コードを使うから必要、という程度のものでした。このため、最初のころは文字を扱う機能を中心に据えたプログラム言語は実験的なものに限られていました。

しかし、事務計算の場合などでは帳票を出力する際には計算した数値といっしょに顧客名や商品名なども打ち出したいわけです。そのため、事務処理言語(COBOLが代表的)を皮切りに、文字を扱う機能がいろいろつけ加えられるようになりました。ただし、当時の汎用の手続き型言語で文字列を扱うのは、文字列の中身を「1つずつ処理する」か、または「ライブラリを呼んで処理させる」ことが標準であり、結構面倒な面がありました。

一方、既に見て来たようにUnixのコマンド群では文字列をパターンで抽出するなど「便利な」ツールが沢山あります。そのため、これらのツールとシェルスクリプトによる文字列処理が多く行われるようになりました。しかしシェルスクリプトはプログラミング言語として見た場合はいろいろ機能が不足しています(もともとフィルタを呼び出すことで多様な処理を行っていました)。

そこでこれを改良して1つの言語内で計算も文字列処理も自由に行えるようにしようとしたのがLarry Wallによって開発されたPerl言語なわけです。Perlはその実用性から広く使われるようになり、これをきっかけにスクリプト言語がさまざまな用途に使える、ということが認識され、スクリプト言語の時代が到来したわけです。ここではテキスト処理の話題なので、以下ではPerlを題材に文字列処理のサンプルを見て行きます。

2.2 Perl入門

前述のように、PerlはLarry Wallによって開発された、近代的スクリプト言語の元祖であり、非常に豊富な機能を持っています。しかしその一方で、全部の機能をマスターしなくても、とりあえず「こう使う」というところだけわかればそれで役に立つという実用的な側面も持っています。Perlの基本的な特徴は次の通りです。

- 変数はシェルのように「\$」をつけて表す。変数は宣言も型指定も不要で、使った時に作られ、数値でも文字列でも自由に入れられる。
- 文字列は「'...'」でも「"..."」でも表せる。前者は完全に「そのまま」の文字列だが、後者はその中に変数があると変数の値が埋め込まれる(実はシェルでもそうなっている)。
- if文やfor文などの本体部分はすべてブロックである必要がある。つまり「{ ... }」で囲む必要がある。

たとえばtest1.plに次のような内容が入っているとします。

```
for($i = 0; $i < 10; ++$i) {  
    print " ", $i;  
}  
print "\n";
```

これを動かすには、perlコマンドを使って次のようにします。

```
% perl test1.pl  
0 1 2 3 4 5 6 7 8 9  
%
```

Perlのprint命令は出力するものをいくつでも指定でき、数値は適当に文字列に変換されて出力されます。また、「\n」(改行文字)を出力しない限りはどんどん続けて同じ行に出力がなされていきます。

2.3 Perlのファイル入出力

次に、Perlでファイルを読み込む方法を見てみましょう。PerlではUnixのファイルディスクリプタに相当するものを「ファイルハンドル」といい、そこから読み込む時には「<名前>」という形を使います。とくに名前を空っぽにした「<>」だと、「プログラム実行にファイル名を指定したときはそのファイルから順に、指定していないときは標準入力から読む」というUnixのフィルタで一般的な動作と同じになります。ですから、次のプログラムで入力をそのまま出力するプログラム、つまりcatと同じプログラムになります。

```
while($line = <>) {  
    print $line;  
}
```

動かしてみましょう(上のプログラムはtest2.plに入っているものとします)。

```
% cat t1  
This is a pen. ← t1の内容  
% perl test2.pl t1 t1 ← t1を2回指定したので  
This is a pen.  
This is a pen. ← 2回連結されている  
%
```

なお、ファイルから読み込んだ時は行末の「"\n"」までくっついた文字列が読み込まれています。

出力については、ファイルハンドルと既に学んだprintを組み合わせ使います。まずopenでファイルを開くと同時にファイルハンドルを作ります。

```
open(F, ">test.txt"); ← 「>」は「出力」をあらわす
```

そのあと、ファイルハンドルを指定したprintを何回でも実行できます(ファイルハンドルの後には「,」は不要なのに注意)。

```
print F "this is a pen.\n";
```

出力が全部終わったら、ファイルを閉じます。これでファイルが完成します。

```
close(F);
```

2.4 フィールドに分かれたデータの処理

先の例ではただ単に入力した行をそのまま出力していましたが、今度は各行の中のデータを処理してみます。いちばんよく使う方法は、splitを使ってデータを複数のフィールドに分けて扱うことです。例として、次のようなデータファイルを扱うものとしましょう:

```
久野 20 180 60  
大木 10 60 170  
吉田 190 100 100
```

何のデータかは分かりませんが、3つの数値はそのデータの「1970年、1980年、1990年」の値だということにして、それぞれの人ごとに平均を計算することにします。

ここで、行が変数\$lineに入っているとして、まずchop(\$line);というのを実行して末尾の改行文字を削除します。次にフィールドに分けるのには、次のようにsplitという関数を使います:

```
($name, $d1, $d2, $d3) = split(/ +/, $line);
```

「/ +/」は「空白1個以上」を表すパターン(正規表現)で、`split`は空白が1個以上あるところで行を分割します。そして分割したそれぞれの部分を左辺の変数`$name`、`$d1`、`$d2`、`$d3`に入れるわけです。なお、データの形式がCSVの場合はパターンに「/,/」を指定すればよいでしょう。

では、平均を計算するプログラムを示します:

```
while($line = <>) {
    chop($line);
    ($name, $d1, $d2, $d3) = split(/ +/, $line);
    $avg = ($d1 + $d2 + $d3) / 3.0;
    print "$name  $d1  $d2  $d3  $avg\n";
}
```

フィールドに分けて、平均を計算して、`print`するだけです。「...」で囲んだ文字列の中に変数を書くと、出力時にその場所に変数の値が埋め込まれます:

```
perl test3.pl t.data
久野  20  180  60  86.66666666666667
大木  10  60   170  80
吉田  190  100  100  130
```

2.5 他形式のファイルの生成

上のような処理だけだったら、もちろん表計算ソフトの方が簡単ですね。せっかく Perl なので、出力をただのテキストにする代わりに、`latex`にしてみましょう。それには `latex` の制御命令を一緒に埋め込んで出力すればいいだけです:

```
print "\\documentclass[12pt,a4j]{jarticle}\n";
print "\\usepackage{graphicx}\n";
print "\\begin{document}\n";
print "\\title{データ表}\n";
print "\\maketitle\n";
print "\\begin{center}\n";
print "\\begin{tabular}{|c|r|r|r|c|}\n";
print "\\hline\n";
print "名前 & 1970 & 1980 & 1990 & 平均\\\\\n";
print "\\hline\n";
$num = 0;
while($line = <>) {
    chop($line);
    ($name, $d1, $d2, $d3) = split(/ +/, $line);
    $avg = ($d1 + $d2 + $d3) / 3.0;
    print "$name & $d1 & $d2 & $d3 & $avg\\\\\n";
    print "\\hline\n";
}
print "\\end{tabular}\n";
print "\\end{center}\n";
print "\\end{document}\n";
```

プログラムの大部分は `latex` のソースを生成するための `print` 命令で、表の各行を処理するループだけが先と同様の計算をおこない、ただし出力するときは各フィールドを&で区切って出力しています。

なお、「\」はエスケープ文字 (特別な処理を行う文字) なので、1つ出力するためには文字列の中では2つ続けて書く必要があります。では実際に動かしてみましょう。

```
perl test4.pl t.data >test.tex
latex test.tex
…以下 latex の回と同様…
```

生成されるソースの中の、表の部分だけを示しておきます:

名前	1970	1980	1990	平均
久野	20	180	60	86.66666666666667
大木	10	60	170	80
吉田	190	100	100	130

2.6 複数のファイルを生成する

もっと頑張って、平均の数値ではなく、グラフを生成してみましょう。それには PostScript 形式でグラフを出力することにします。PostScript については既にやりましたが、今回使用する5つの命令を再掲しておきます。

- `newpath` — 新しい線引きを開始する。
- `X Y moveto` — ペンを指定した座標 (X, Y) に移動。
- `X Y lineto` — 座標 (X, Y) までの線を登録しながら移動。
- `stroke` — `lineto` で指定した線引き一式を実行する。
- `W setlinewidth` — 線の太さを W pt にする。

以下のプログラムでは、個人ごとに 300×200 pt ($1\text{pt} = \frac{1}{72}\text{inch}$) の描画面を用意し、そこに上の操作を使ってグラフを描きます。実際にはこれは1人に1つずつ、`fig1.ps`のような名前の PostScript ファイルを開いて、そこに命令を書き込んでいます。latex の中では次のような命令を使ってそのファイルを埋め込むことができます。

```
\includegraphics[scale=倍率]{ファイル名}
```

プログラムを見てみましょう (「.」は文字列連結の操作を表します):

```
print "\\documentclass[12pt,a4j]{jarticle}\n";
print "\\usepackage{graphicx}\n";
print "\\begin{document}\n";
print "\\title{データ表+グラフ}\n";
print "\\maketitle\n";
print "\\begin{center}\n";
print "\\begin{tabular}{|c|r|r|r|c|}\n";
print "\\hline\n";
print "名前 & 1970 & 1980 & 1990 & グラフ\\\\\n";
print "\\hline\n";
$num = 0;
while($line = <>) {
    chop($line);
    ($name, $d1, $d2, $d3) = split(/ +/, $line);
    $file = "fig" . ++$num . ".ps";
```

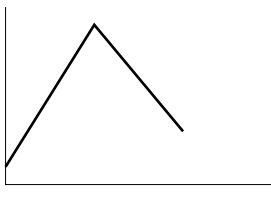
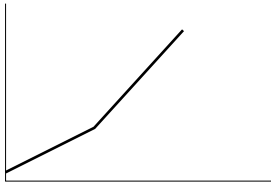
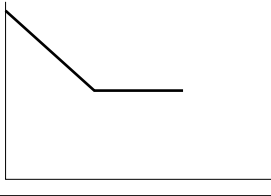


```

open(F, ">".$file);
print F "%!PS-Adobe-2.0\n";
print F "%BoundingBox: -5 -5 310 210\n";
print F "newpath 0 200 moveto 0 0 lineto 300 0 lineto stroke\n";
print F "3 setlinewidth\n";
print F "newpath 0 $d1 moveto 100 $d2 lineto 200 $d3 lineto stroke\n";
close(F);
print "$name & $d1 & $d2 & $d3 &";
print "\\includegraphics[scale=0.333]{$file}\\\\\\n";
print "\\hline\n";
}
print "\\end{tabular}\n";
print "\\end{center}\n";
print "\\end{document}\n";

```

変数`$file`にPostScriptのファイル名を生成し、それを出力指定で`open`し、そこにPostScriptを書き出して`close`します。latexの側では平均は計算せず、代わりに`includegraphics`で生成したPostScriptファイルを埋め込みます。今度は表の部分は次のようになります:

名前	1970	1980	1990	グラフ
久野	20	180	60	
大木	10	60	170	
吉田	190	100	100	

演習 9-2 LaTeX を生成する例題の Perl プログラム (2 種類) をコピーしてきて動かしてみなさい。`t.data` の内容は適宜変更してみる。うまく動いたら、次の課題から 1 つ以上 (できれば全部) 選んでやってみなさい。

- 表を生成する例題を直して、平均に加えて標準偏差も表に記載されるようにしなさい。⁸
- グラフを生成する例題を直して、グラフの中の平均の位置に水平線を引くようにしなさい。
- このデータファイルの内容をより分かりやすく表現する図的表現を考え、グラフの代わりにその図的表現を生成するようになささい (データ形式を変更してもよいです)。

「◎」の条件: (1) 小問 2 つ以上やってあり、(2) やったことの説明やそれに対する考察がきちんと書かれていること。

⁸Perl では「平方根」は `sqrt(x)` で計算できます。

3 PHP と Web サーバ側プログラミング

3.1 PHP 言語とその由来

前回は取り上げたように、Web アプリケーションは基本的には HTML で記述されたフォームによるユーザインタフェースと、そこから提出されてきたデータを処理する、サーバ側の CGI プログラムとの組み合わせでできています (今日では Ajax などもっと込み入った形態のものもありますが)。

CGI プログラムとは、CGI(Common Gateway Interface) と呼ばれる規約に従って動作するプログラムのことで、この規約が Web サーバとプログラム、Web ページとプログラムの間のやりとり方法を定めています。CGI プログラムは、WWW の初期には C 言語など普通のプログラミング言語で書かれたり、シェルスクリプトを活用して作られたりしてきましたが、その性質上、文字列の処理を多く必要とし、Perl などのスクリプト言語との相性がよいため、現在では Perl をはじめとするスクリプト言語で作られることが多くなっています。

そのような言語のひとつ、**PHP**(Hypertext Processor — どこに最初の P があるんだろうと思うでしょうが、その初期において Personal Home Page tools と呼ばれていたことに由来するらしい) は、Web サーバに埋め込んで実行するスクリプトを書くためのプログラミング言語として、1995 年に Rasmus Lerdorf によって開発されました。その後開発者も実装も変遷して、現在は 2004 年に公開された PHP5 が最新版となっており、以下の説明もこれを対象とします。

PHP の最大のアイデアは、HTML を記述する中に PHP 言語のプログラムが埋め込まれてサーバ上で動くという方式を確立したことで、現在では MS ASP(Active Server Page)、JSP(Java Server Page) など多くのものがそのマネをしています。もう 1 つの特徴は豊富なモジュールによってインターネット上のさまざまな標準をサポートしていることで、そのために PHP のマニュアルは (そして解説本も) ぶ厚いものとなっています。

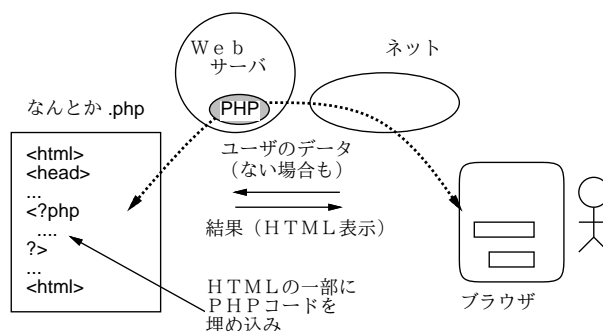


図 2: PHP によるデータの受け取りと処理

さらに、現在標準の Web サーバとして広く使われている Apache ではモジュールとして PHP を組み込むことができ、「.php」で終わるファイルに対しては自動的に PHP 処理系が解釈実行するように設定できます。この場合、「.php」で終わるファイルは自動的に PHP 処理系が実行し、これによって (PHP 言語で記述された) ユーザデータの処理やページ内容の処理による生成が可能になります (図 2)。我々の内部サーバもこのように設定してあります。

一般に、ユーザ側から PHP プログラムを参照する形は次のどちらかの方法になります。

- 「なんとか.php」の URL を自分で開いたりリンクを通じて開く — データは送信しないので、PHP 側では「最初に開かれた」ものとして処理を行う。
- フォーム要素の action 属性として「なんとか.php」の URL が指定されていてそこに送信する — データが送られるので、PHP 側では「送られたデータを処理し、新たな (結果の) ページを返送する」ものとして処理を行う。

以下では、PHP 言語を使うとどのように簡単に CGI プログラム、つまり Web アプリケーションのサーバ側プログラムが構成できるか、ということを見物して行きたいと思います。

3.2 PHP 言語の基本的な構成

最初に知っておくべきことは、Web で使用する場合の PHP プログラムは HTML ファイルの中に次のような形 (処理命令と呼ばれるものの一種) として埋め込む、ということです。

```
<?php …PHP プログラム… ?>
```

なお、PHP プログラムの部分は何行に渡っても構いません。PHP プログラムの中では C や C++ や Java と同じコメント (`/* ... */` で囲むコメント、または `//` (`#` でもよい) から行末までのコメントが使えます。

もう 1 つ重要なことですが、PHP プログラムの中で HTTP ヘッダを出力する機能 (`header()` 関数) を使うときは、一切の通常出力を行う前に行う必要があることです。たとえば、PHP で日本語が正しく表示できるように文字エンコーディングを指定するときにも `header()` が必要です。なので、PHP プログラムをサーバ上に置く時は次の形のものを使うとよいでしょう (文字化けの問題がないなら、最初のヘッダ生成などは省略してもよいです):

```
<?php
    header("Content-type: text/html; charset=euc-jp");
    /* その他ヘッダ出力はここで */
?>
<!DOCTYPE html>
<html><head><title>title…</title>
<style type="text/css">
/* CSS 記述はここに */
</style><head><body>
<?php
    /* ページ本体出力を含む動作 */
?>
</body></html>
```

ファイル中に日本語を含める場合は、ファイルの文字コードは EUC にしてください。Emacs であれば「Ctrl-X [RET] f euc-jp [RET]」を実行することで編集集中のファイルの文字コードを EUC にできます。ステータス表示の最後が「…E」となっていれば EUC ですので、そうなっていない時だけ上記を実行すればよいでしょう。

ではまず最初に、PHP で単純に HTML を生成するという簡単なサンプルを示します (図 3)。

```
<php? header("Content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE html>
<html><head><title>php sample</title>
<style type="text/css">
div.num { margin: 2px 20px; background: rgb(200,215,255) }
</style></head><body>
<h1>数を並べる</h1>
<?php
    for($i = 0; $i < 10; ++$i) {
        echo "<div class='num'>番号{$i}です。</div>";
    }
?>
</body></html>
```

資料では見やすいように下げてありますが、実際のファイルでは最初の「<?php」より前に 1 文字の空白もあってはいけません (あるとページ内容が出力されてしまい、`header()` が失敗します)。



図 3: PHP による簡単なページ生成

この例は、単に「番号 i です。」という div 要素が 10 個出力されるだけです。ソースと対比すればプログラムが動いて出力していることに納得が行くと思います。echo 命令は文字列を出力する機能で、文字列中に変数を埋め込む場合は (Perl とはちょっと違って) 「 $\{ \$変数名 \}$ 」という形を使います。また、CSS を使って div 要素のマージンと背景色を指定することで、ページの見た目をそれらしくしています。

演習 9-3 「10 未満の数を表示する」PHP プログラムを GSSM 内部サーバの自分のページとして設置し、動作を確認しなさい。動いたら次の課題から 1 つ以上 (できれば全部) 選んでやってみなさい。

- a. 10000 未満の数を表示するように修正する。できれば、縦に並ぶと長いので見やすいように工夫するとなおよい。
- b. 掛け算の九九の表を表示するようにする。できれば、HTML の表を使って縦横にきれいに並んだものにできるとなおよい。
- c. Web ページで使えるさまざまな色の「色見本」のページを表示するようにする。RGB 値は 0~255 段階なので、たとえば 16 飛びに変化させて全部の組み合わせを見せるなどが考えられる。⁹ できれば、ぱっと見て好きな色を見つけやすいように工夫するとなおよい。

「◎」の条件: (1) 小問 2 つ以上に回答してあり、(2) いずれも「できれば」までやってあり、(3) やったことに対する説明やそれに対する考察が適切であること。

3.3 フォームデータの受け取り

PHP から HTML フォームのデータを受け取るのは非常に簡単で、form 要素からの送信メソッドが get のときは「 $\$_GET['名前']$ 」、post のときは「 $\$_POST['名前']$ 」を参照するだけです (「名前」は form 内の入力部品の name 属性で指定した名前を指定します)。

また、関数 `isset()` を使ってその名前のデータが送られているかどうかを判定することもできます。これを利用して、「最初に呼び出された場合」と「データが送られて来た場合」を次のように枝分かれして扱うのが普通です (「名前」にはフォームで指定した名前の 1 つを指定すればよいでしょう)。

```
<?php
    if (isset($_GET['名前'])) {
```

⁹ ヒント: 任意の HTML 要素についてタグ内に「`style="background:rgb(赤, 緑, 青)"`」という属性をつければその指定した色に塗ることができます。

```

        フォームからのデータがある場合の処理
    } else {
        最初にページが表示される場合の処理
    }
?>
...
<form action="#">
...
</form> ←送信先として自分を指定

```



図 4: PHP によるフォームデータの受け取り

では、先の例を手直ししてフォームにより等差数列の初項、階差、上限を指定できるようにしてみます。

```

<php? header("Content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE html>
<html><head><title>php sample</title>
<style type="text/css">
div.num { margin: 2px 0px; background: rgb(200,215,255) }
form { padding: 1ex; background: rgb(200,185,220) }
</style></head><body>
<h1>数を並べる 2</h1>
<div><form action="#">
<?php
    if(isset($_GET['init'])) {
        $i = $_GET['init']; $s = $_GET['step']; $m = $_GET['max'];
    } else {
        $i = 0; $s = 1; $m = 10;
    }
    echo " 初項:<input type='text' name='init' size='4' value='{ $i }'>";
    echo " 階差:<input type='text' name='step' size='4' value='{ $s }'>";
    echo " 上限:<input type='text' name='max' size='4' value='{ $m }'>";
    echo " <button>変更</button></form></div>";
    for($k = $i; $k < $m; $k = $k + $s) {
        echo "<div class='num'>番号{ $k }です。</div>";
    }

```

```

}
?>
</body></html>

```

この例ではフォームの入力部品として、text 入力欄を 3 つと送信ボタンを用意しています。text 入力欄部分の出力を PHP 側で行っているのは、入力欄の value 属性に現在値を入れた形で出力したいためです。

3.4 共有データとファイル入出力

先の例ではデータを PHP で処理はしていましたが、各ユーザが入力したデータに対して処理をして返すだけでした。実際の Web アプリケーションの本質は、サーバ上のデータを読み書きすることで、複数のユーザがデータを共有できる点にあります。その簡単な例として「掲示板」のようなものを作ってみましょう (図 5)。



図 5: PHP による掲示板

具体的には、掲示板に書き込むメッセージを PHP ソースと同じ場所にある msg.txt というファイルに蓄積していきます。このファイルは最初から存在している必要があり、また Web サーバのプロセス (ユーザ ID nobody で動作している) に読み書きできる必要があるため、次のようにして空 (実際には改行 1 バイトだけ) の内容を用意し、保護モードも「誰でも読み書き可能」にしておきます。

```

% echo ' ' >msg.txt
% chmod a+rw mgs.txt

```

ファイルを読み書きする PHP の関数として、以下では次のものを使います (file() だけは後の例で使います)。

- fopen(ファイル名, モード) — ファイルを開く (読み書きの準備をする)。モードは「r」なら読み込み、「w」なら書き出し、「a」なら追加書き込みを意味する。この関数はストリーム (読み書きチャンネル) を返すので、それを指定して以後の読み書きを行う。
- fclose(チャンネル) — チャンネルを閉じる (対応するファイルの読み書きを終了する)。
- fwrite(チャンネル, 文字列) — チャンネルを指定して文字列を書き込む。
- fread(チャンネル, バイト数) — チャンネルを指定して指定バイト数だけ読み込み、内容を文字列として返す。
- filesize(ファイル名) — ファイルの大きさ (バイト数) を取得する。

- file(ファイル名) — ファイルの内容を 1 行 1 要素の配列として返す。

では掲示板プログラムを見てみましょう。冒頭でフォームの書き込みメッセージがあるかどうか調べ、ある場合はファイルを追記モードで開いて末尾に名前とメッセージを入れ、チャンネルを閉じます。その後フォームを書き、最後にファイルの内容を全部読んで出力します。このようにすることで、初回でも書き込み時でも処理がほぼ共通になります。

```
<?php header("content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE html>
<html><head><title>php sample</title>
<style type="text/css">
div.msg { margin: 2px 0px; padding: 4px; background: rgb(200,215,255) }
form { padding: 1ex; background: rgb(200,185,220) }
</style></head><body>
<h1>掲示板</h1>
<?php
    if(isset($_POST['msg'])) {
        $fd = fopen('msg.txt', 'a');
        fwrite($fd, "<div class='msg'>{$_POST['name']}: {$_POST['msg']}</div>\n");
        fclose($fd);
    }
    $fd = fopen('msg.txt', 'r');
    echo fread($fd, filesize('msg.txt'));
    fclose($fd);
?>
<div><form method="post" action="#">
名前: <input type="text" name="name"><br>
<textarea name="msg" rows="5" cols="40"></textarea> <button>書き込み</button>
</div></form>
</body></html>
```

ただし、このようなファイル共有型のアプリケーションには注意すべき点があります (図 6)。それは、Web アプリケーションは多数のユーザが Web サーバにアクセスしてきてて並行して実行されるという点です。このため、複数行に渡るファイルアクセスのような処理は、複数ユーザの処理が「重なって」実行されてしまい、正しくないデータが書かれてしまう可能性があるという点です。これを避ける方法としては、次のものがあります。

- (a) 干渉しないような形でファイルアクセスを行う。
- (b) ファイルにロックを掛ける。
- (c) ファイルの代わりにデータベースシステムを使う。

今回の例題では、(a)の方法を用いています。すなわち、各ユーザの処理は「書き込み」と「読み出し」に別れていて、書き込みの処理は数十バイト程度のデータを追加書き込みするだけです。この程度の量なら 1 つのシステムコールで書かれるため、他のプロセスの処理と互い違いになる心配はほとんどありません (あったとしたも 1 個ぶん書き込みが失われる程度ですが、その場合はユーザにもう 1 度書いてもらってもそんなにまずくないでしょう)。

(b)については、さほどアクセスが多くないサーバでの処理ならいいのですが、ファイルをロックしてしまうと他のプロセスがファイルを使えるまで待たされるので、大量にアクセスがあると多くのプロセスが待たされてサーバに負荷が掛かるという弱点があります。そのため、多くのデータを扱い多数のユーザが共有する場合は (c) データベースシステムを利用する方法が一般的です。データベー

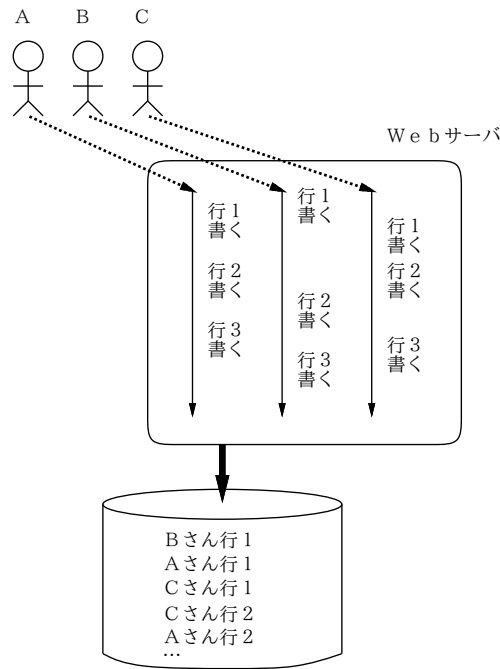


図 6: 並行アクセスの問題

システムでは複数ユーザが並行してデータを処理することが基本的な要請なので、そのための機構が充実していて、性能上も問題ないように作られているからです (このあたりは次回扱います)。

演習 9-4 「等差数列を表示する」および「掲示板」PHP プログラムを GSSM 内部サーバの自分のページとして設置し、動作を確認しなさい。動いたら次の課題から 1 つ以上 (できれば全部) 選んでやってみなさい。

- a. 初項・公比・上限を指定して等比数列を表示するようにする。
- b. 借入れ金額・毎年返済額・金利を指定して元利均等返済ローンの毎年の返済状況を表示する (一定年数返済しても終わらなければ打ち切る)。
- c. その他、自分で面白いと思うページを作ってみる。

「◎」の条件: (1) 小問 2 つ以上に回答してあり、(2) やったことの説明やそれに対する考察が適切であること。

4 JavaScript と Web クライアント側プログラミング

4.1 JavaScript とブラウザ

前節までで説明した PHP がサーバ上で動く「サーバ側プログラミング」の機能を提供しているのに対し、JavaScript は HTML と一緒にブラウザに取り込まれ、ブラウザ上で動く「クライアント側プログラミング」の機能を提供しています (厳密に言えば JavaScript 言語をブラウザ上以外で使う例もありますが、現実に存在し使われている大多数の JavaScript 処理系はブラウザ上のものです)。

そして、JavaScript コードはブラウザ上で動くことで、対応する HTML によるページを表示している「間だけ」ブラウザの動作を調整したり、さまざまなアプリケーションに必要な動作を行ったりできます。

なぜ「間だけ」なのでしょう? それは、あるページを見おわって別のページに行っても前のページの JavaScript の効果が残っているようだと、それぞれのページの JavaScript プログラムが干渉して

しまいますし、それに読み手のプライバシーを侵害するような危険なスクリプトが作れてしまう可能性が生じるからです。

そもそも別のページ云々とは独立に、JavaScript でブラウザを制御できる範囲はユーザにとってプライバシー侵害やセキュリティ上の危険が生じないように注意深く限定されています。このような安全性の保証は、Web 上にあつてブラウザにダウンロードされてきて動くコードすべてに対して共通に課せられる課題だと言えます。(にもかかわらず、この種のコードによるセキュリティホールは今日でもしばしば問題になっています。)

具体的にブラウザ上の JavaScript でできることを整理すると次のようになります。

- ブラウザ自体の動作の制御 — 表示ページの切り替え、1つ前のページに戻る、窓サイズの変更、新しい窓を開く、ダイアログを出すなど。
- イベントへの応答 — ボタンや任意の要素をクリックしたりその上にマウスポインタを置いた時に動作を起動できる (の要素の開始タグをクリックなら「onclick="コード"」、マウスポインタを置いた場合なら「onmouseover="コード"」などと指定するとその JavaScript コードが動作する)。
- フォーム部品の変数の参照と書き換え — フォーム部品 (GUI 部品) の値を読み取ったり、その値を変更できる。
- ページ内容の書き出し (読み込み時) — スクリプト内で `document.write(文字列)` を呼び出すことにより、そのスクリプトのある位置に任意の HTML を書き出すことができる。ただしこの機能が使えるのはページ読み込み途中に限られる。
- ページ内容の変更 (読み込み終了後) — ページを読み終わった後でも、後述する DOM または innerHTML インタフェースを使うことで任意のページ内容を書き換えることができる。
- ページ要素のスタイル変更 — 任意の HTML 要素に対して CSS スタイル指定を設定することで色や位置などを変更できる。
- 一定時間後/一定時間間隔での動作実行 — 一定時間後に動作を起動したり、一定間隔で動作を行いアニメーションのような効果を持たせることができる。

4.2 例題: 摂氏華氏変換

では簡単な例として、摂氏と華氏の温度を相互変換するページを作ってみましょう (図 7)。ここでは、温度を入力する入力欄、結果を表示する入力 (?) 欄、変換の方向 (摂氏→華氏、華氏→摂氏) を選択する選択メニュー、計算ボタンを GUI 部品として用意しています。変換の計算式は次の通です。

$$c = \frac{5}{9}(f - 32), \quad f = \frac{9}{5}c + 32$$



図 7: JavaScript を用いた温度変換

JavaScript コードと GUI 部品定義を含んだ HTML を以下に示します。計算ボタンの onclick ハンドラで JavaScript の関数 calc() を呼び出し、この中で入力欄の値を数値として取り出して計算式に従って計算し、表示欄に書き込んでいます。関数 parseFloat() は文字列を数値に変換する組み込みの関数です。

```
<!DOCTYPE html>
<html><head>
<title>sample</title>
<style type="text/css">
form { background: rgb(200,255,235); padding: 1em }
</style>
<script type="text/javascript">
function calc() {
  var i = parseFloat(document.forms.data.elements.itemp.value);
  if(document.forms.data.elements.dir.selectedIndex == 0) {
    var o = (9/5)*i + 32;
  } else {
    var o = (5/9)*(i - 32);
  }
  document.forms.data.elements.otemp.value = o;
}
</script>
</head><body>
<h1>温度変換</h1>
<div><form name="data" onsubmit="return false">
  入力温度: <input type="text" name="itemp" size="5" value="0">
  換算温度: <input type="text" name="otemp" size="20"><br><br>
  <select name="dir">
  <option>摂氏→華氏</option><option>華氏→摂氏</option></select>
  <button onclick="calc()">計算</button>
</form></div></body></html>
```

このコードは、「変換」ボタンを押した時はじめて計算が行われます。このようなモデルは古典的ですが、明確で分かりやすいという利点があります。

しかしもっと今風に工夫するなら、手元で計算しているのですから、ユーザが1文字変更するごとにそれに対応する結果を表示してもいいはずです。ついでに、摂氏→華氏、華氏→摂氏のどちらなのかも同じ入力欄で指定させられますね (図 8)。

これを行うコードは次のようになっています:

```
<!DOCTYPE html>
<html><head>
<title>sample</title>
<style type="text/css">
div { background: rgb(200,255,235); padding: 1em }
</style>
<script type="text/javascript">
function chg(elt) {
  var c = elt.value.charAt(0);
  var i = parseFloat(elt.value.substring(1));
  if(c == 'F' || c == 'f') {
```

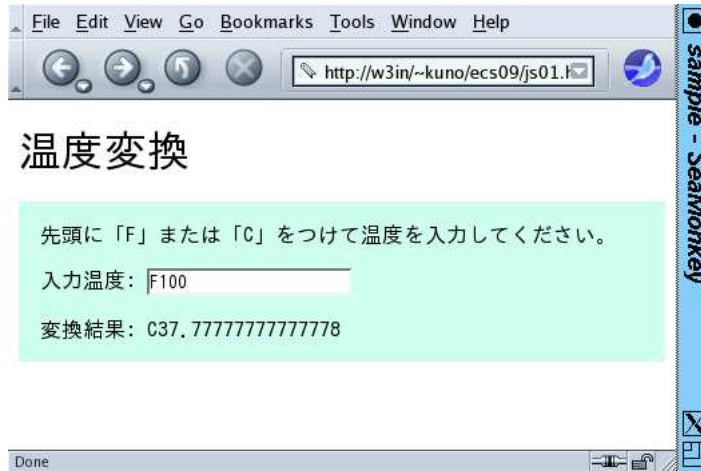


図 8: 摂氏華氏変換 (入力即時変換)

```

    document.getElementById('s0').innerHTML = 'C' + ((5/9)*(i - 32));
  } else {
    document.getElementById('s0').innerHTML = 'F' + ((9/5)*i + 32);
  }
}
</script>
</head><body>
<h1>温度変換</h1>
<div>先頭に「F」または「C」をつけて温度を入力してください。 <br><br>
入力温度: <input type="text" name="it" value="F" onkeyup="chg(this)"><br><br>
変換結果: <span id="s0">???</div></body></html>

```

まず、先の例と違って form 要素を使っていません。そのかわり、入力欄 (`input type="text"`) に `onkeyup` ハンドラが指定してあり、そこで `this` をパラメタとして関数 `chg()` を呼んでいます。実は `this` はハンドラを持っている HTML 要素 (`input`) のオブジェクトを参照していて、このため関数 `chg()` の中で渡された引数の属性 `.value` を参照することで入力文字列が取れます。 `onkeyup` だと打鍵 1 回ごとにハンドラが呼ばれるので、その中で先頭の文字が F か C かに基づいて適切な変換を行い、最後に `div` 要素の内側にその結果を書き込みます。 `div` 要素を参照するには、 `id` を指定して `document.getElementById()` を呼びます。中身を書き換えるのは、 `.innerHTML` プロパティに書き込めばできます。

このように、HTML 上で表されている各要素は JavaScript 側からはオブジェクトとして取り扱え、そのプロパティやメソッドを用いてさまざまな変更を直接施すことができるわけです。これをどのブラウザでも同じように行わせるため、HTML や XML などの文書 (ドキュメント) をどのようなオブジェクトの構造に対応させるかの標準があり、DOM (Document Object Model) と呼ばれています。

もう 1 つの例として、今度は「スライダを動かすことでそれに対応する摂氏と華氏の温度を表示させる」ようなインタフェースを作ってみました (図 9):

```

<!DOCTYPE HTML html>
<html><head>
<title>sample</title>
<style type="text/css">
div { background: rgb(200,255,235); padding: 1em }
#d0 { background: rgb(10,200,190); position: absolute;

```



図 9: スライダを使った温度変換

```

    top: 200px; left: 20px; width: 400px; height: 10px }
#s0 { color: red; position: absolute; top: 5px; left: 195px }
</style>
<script type="text/javascript">
function drag(ev) {
    var x = ev.clientX || ev.pageX;
    x = x - 25; if(x < 0) x = 0; if(x > 400) x = 400;
    document.getElementById('s0').style.left = x + 'px';
    var c = 0.5*x - 50, f = (9/5)*c + 32;
    document.getElementById('d1').innerHTML = '摂氏 = ' + c;
    document.getElementById('d2').innerHTML = '華氏 = ' + f;
}
</script>
</head><body>
<h1>温度変換</h1>
<div id="d0" onmousemove="drag(event)"><span id="s0">▲</span></div>
<div id="d1">?</div><div id="d2">?</div></body></html>

```

コードの説明ですが、スライダ全体とその中の「つまみ」をそれぞれ div 要素、span 要素として用意し、CSS で位置を絶対配置 (`position: absolute`) に指定します。こうしておくことで、`left`、`top`、`width`、`height` などを設定することで要素の位置と大きさを自由に制御できます。そして、HTML 側ではスライダ部分に `onmousemove` を指定して、そこでイベントオブジェクトをパラメタとして関数 `drag()` を呼び出します。この中では、イベントオブジェクトのプロパティ `.clientX` または `.pageX` (ブラウザの種類で違います) を参照することでマウスの X 座標を取り出し、それに基づいて「つまみ」の位置を変更するとともに、その位置に対応した摂氏と華氏の温度を表示します。位置の変更などは、HTML 要素オブジェクトの `.style` プロパティの子プロパティを設定することで、CSS で設定するのと同様の設定が自由に行えることを利用しています。

このように、JavaScript ではイベントハンドラと位置の制御などを行うことで、さまざまな柔軟なインタフェースを作れるようになっているわけです。

演習 9-5 JavaScript と HTML の GUI 部品を組み合わせる次のような処理をするページを 1 つ以上 (できれば全部) 作ってみなさい。

- a. 次々に値を入力してその累計を計算できるページ。できれば、累計をクリアする機能もあるとなおい。

- b. N を指定すると 2 の N 乗を計算できるページ。できれば、 2 以外の値の N 乗も計算できるとなおよい。
- c. その他、自分で役に立つ/面白いと思う計算をしてくれるページ。できればユーザインタフェースに工夫があるとなおよい。

「◎」の条件: (1) 小問2つ以上に回答しており、(2) いずれも「できれば」の部分までやってあり、(3) やったことの説明やそれに対する考察がきちんとしていること。

演習 9-6 JavaScript と HTML を使って、自分独自の温度変換インタフェースあるいはそれ以外の処理を行うインタフェースを実装してみなさい。「◎」の条件: どのような独自性なのか、実装して試してみてどのようなことが分かったかについて、きちんと書かれていること。

5 まとめ

スクリプト言語とは「簡単にプログラムが書ける言語」という意味あい、最初はシェルスクリプトなど簡便な言語が使われていましたが、次第に各種用途に使える高機能なものが普及してきました。今回はシェルスクリプト、テキスト処理に多く使われる Perl、Web のサーバ側プログラミングに使われる PHP、Web のクライアント側プログラミングに使われる JavaScript の4つを取り上げ見ました。スクリプト言語で色々なことができることがお分かり頂けたと思います。