誰にでも書ける #! /bin/nawk -f 講座
第 1 回
「サイン、コサイン、タンジェント」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : awk でサインが使えないんです。
root : サインって？
yasuoka : サイン、コサイン、タンジェントのサイン。
root : ああ、そのサインか。
yasuoka : そう、そのサインを awk で使ってみたんですけど

```
~/bin% cat sin （ぽこ）
#! /bin/awk -f
# "sin" Version 1.0
BEGIN{
  printf("degree? ");
}
{
  if($0=="")
    exit;
  printf("%f\ndegree? ",sin($1));
}
~/bin% sin （ぽこ）
degree? 
```

結果が全然変なんです。

```
degree? 45 （ぽこ）
45.0000
degree? 30 （ぽこ）
30.0000
degree? （ぽこ）
~/bin% 
```

root : sin は awk じゃ使えないよ。
yasuoka : えー、でも使えるって書いてある本があったんですけど。
root : その本はたぶん、 nawk のことを書いてるんじゃないかな。
yasuoka : nawkって何ですか？
root : awk の新しいヴァージョンだよ。古い awk と互換性があって、なおかつ新しい機能が追加されてる。
yasuoka : どうやって使うんですか？
root : このスクリプトの 1 行目を #! /bin/nawk -f に書き換えればいいよ。ただマシンによっては、 /bin/nawk じゃなくて /usr/local/bin/gawk に新しい awk があったりするから、その辺は注意が必要だね。
yasuoka : そうなんですか。

（間）

yasuoka : できました。

```
~/bin% cat sin （ぽこ）
#! /bin/nawk -f
# "sin" Version 1.1
BEGIN{
  printf("degree? ");
}
{
  if($0=="")
    exit;
  printf("%f\ndegree? ",sin($1));
}
~/bin% sin （ぽこ）
degree? 
```

これでどうかな？

```
degree? 45 （ぽこ）
0.850904
degree? 30 （ぽこ）
-0.988032
degree? 
```

あれ、やっぱり答が変。

root : nawk の sin はラジアンだからね。
yasuoka : あ、そうなんですか。じゃあ、ちょっと書き換えないと。

（間）

yasuoka : 今度こそどうかな？

```
~/bin% cat sin （ぽこ）
#! /bin/nawk -f
# "sin" Version 1.2
BEGIN{
  printf("degree? ");
}
{
  if($0=="")
    exit;
  printf("%f\ndegree? ",sin($1*0.0174532925));
}
~/bin% sin （ぽこ）
degree? 45 （ぽこ）
0.707107
degree? 30 （ぽこ）
0.500000
degree? 
```

お、うまくいった。
root : なかなかやるな。
yasuoka : へへー。

```
degree? （ぽこ）
~/bin% 
```

ところで root さん。
root : 何だい？
yasuoka : nawk では sin 以外の新しい機能はないんですか？
root : 実はたくさんの機能が追加されてるんだ。まずは普通の式からいこうか。

文字列	文字列（複数並べてもよい）
数	10 進数（小数も可）
式+式	2 式の和
式-式	左式から右式を引いた差
式*式	2 式の積
式/式	左式を右式で割った商
式%式	左式を右式で割った余り
式^式	左式の右式乗
式==式	2 式が等しいとき 1、さもなくば 0
式!=式	2 式が等しくないとき 1、さもなくば 0
式>式	左式が右式より大きいとき 1、さもなくば 0
式>=式	左式が右式以上のとき 1、さもなくば 0
式<式	左式が右式未満のとき 1、さもなくば 0
式<=式	左式が右式以下のとき 1、さもなくば 0
文字列~/正規表現/	文字列が正規表現にマッチするなら 1、しないなら 0
文字列~文字列	右文字列を正規表現とみなし、他は上に同じ
文字列!~/正規表現/	文字列が正規表現にマッチしないなら 1、するなら 0
文字列!~文字列	右文字列を正規表現とみなし、他は上に同じ
!式	式が 0 あるいはヌルのとき 1、さもなくば 0
式&&式	2 式のどちらかが 0 かヌルのとき 0、さもなくば 1
式||式	2 式の両方が 0 かヌルのとき 0、さもなくば 1
式?式:式	左式が 0 かヌルのとき右式、さもなくば中式
int(式)	式の整数部
log(式)	式の自然対数
exp(式)	e の「式」乗
sqrt(式)	式の平方根
sin(式)	式の正弦
cos(式)	式の余弦
atan2(式,式)	左式を垂辺、右式を底辺とする場合の逆正接
rand()	0 以上 1 未満の乱数
length(文字列)	文字列の長さ
substr(文字列,式)	文字列の「式」文字目以降
substr(文字列,式,式)	文字列の左「式」文字目から右「式」文字
index(文字列,文字列)	左文字列の何文字目に右文字列を含むか（ない時 0）
sprintf(フォーマット)	フォーマットにしたがった文字列
(式)	優先度を変える

yasuoka ： awk では条件にしか使えなかったものも、全部普通の式になっちゃったんですね。

root ： そういうこと。あと三角関数と乱数なんかが増えてる。あ、乱数は使う前に srand() が必要だけどね。

srand(式);
　「式」を rand() で発生する乱数の種とする。
srand();
　現在の時刻を乱数の種とする。

yasuoka ： 正規表現は変わったんですか？

root ： いや、変わってない。

^	文字列の先頭
$	文字列の末尾
.	任意の 1 文字
[複数の文字]	複数の文字のいずれか 1 文字、 - は省略を意味
[^複数の文字]	複数の文字以外の 1 文字、上と同じ省略が可能
*	直前の正規表現の 0 回以上の繰り返し
+	直前の正規表現の 1 回以上の繰り返し
?	直前の正規表現の 1 回以下の繰り返し
正規表現|正規表現	いずれかの正規表現
(正規表現)	正規表現それ自身、 * や | のおよぶ範囲を限定
\n	改行コード
\t	タブ
\^	^
\$	$
\.	.
\[	[
\]	]
*	*
\+	+
\?	?
\\|	|
\(	(
\)	)
\/	/
\\	\
その他の文字	その文字自身

yasuoka ： printf は？

root ： 変わってない。

printf(フォーマット);
　フォーマットにしたがって、文字列を標準出力に出力する。フォーマットは以下の形である。
　　文字列,式,式…
　式の値によって、文字列中のフォーマット制御文字列が置き換えられる。式の個数は、文字列中のフォーマット制御文字列の個数と、一致していなければならない。
フォーマット制御文字列は、以下の通り。

%c	数に対応する ASCII 文字
%d	10 進整数
%f	小数部 6 桁の 10 進数
%.数f	小数部「数」桁の 10 進数（数は自然数のみ）
%o	8 進整数
%s	文字列
%.数s	「数」文字を越えない文字列（数は自然数のみ）
%x	16 進整数

いずれも、 % の後に整数を書くことにより、桁数を指定できる。例えば、 %12d と指定すると、これに対応する値が 12 桁未満の時には、値の前に空白が詰まって 12 文字分となる。 %-12d と指定すると、対応する値が 12 桁未満の時には、値の後に空白が詰まって 12 文字分となる。 %012d では、値の前に 0 が詰まって 12 文字分となる。
特殊文字のための制御文字列は、以下の通り。

%%	%
\n	改行コード
\t	タブ
\\	\

\n、 \t、 \\ は通常の文字列中にも使用できる。

root ： でも if は少し変わった。

```
if(式)
  命令;
else
  命令;
```

式が 0 でもヌルでもなければ直後の命令を、さもなくば else の命令を実行する。 else 節はなくてもよい。

yasuoka : 条件のところが、単なる式になったんですね。 if の直後の命令は、 1 つしか書けないんですか？

root : いや、 { } を使えば、複数書くこともできる。

```
{
  命令;
  命令;
  …
}
```

複数の命令をひとまとめにする。

root : このスクリプトだと関係ないけど、実は exit も少し変わってる。

```
exit(式);
```

式の値をエグジットステイタスとして、実行を終了する。

```
exit;
```

実行を終了する。それ以前に exit(式); が実行されていない場合、エグジットステイタスは 0。

END{以外での exit は、 END{を実行してから終了する。

yasuoka : BEGIN{での exit は END{を通るようになったんですね。

root : そういうこと。

yasuoka : うーむ。

```
~/bin% cat sin （ぽこ）
#! /bin/nawk -f
# "sin" Version 1.2
BEGIN{
  printf("degree? ");
}
{
  if($0=="")
    exit;
  printf("%f\ndegree? ",sin($1*0.0174532925));
}
~/bin% 
```

awk のスクリプトはそのまま nawk で動くんですか？

root : 普通はね。何か試してごらん。

yasuoka : えっと確か、前に書いたスクリプトに

```
~/bin% cat yasuoka （ぽこ）
#! /bin/awk -f
# "yasuoka" Version 2.0
{
  t=$0;
  while(i=index(t,"yasuoka"))
    t=substr(t,1,i-1) "YASUOKA" substr(t,i+7);
  printf("%s\n",t);
}
~/bin% 
```

あ、これだ、これだ。

root : 入力の yasuoka を大文字にするスクリプトだね。

yasuoka : そうです。これを nawk のスクリプトにするには、どうすればいいんですか？

root : 1 行目を #! /bin/nawk -f に書き換えるだけでいいよ。

（間）

yasuoka : できました。

```
~/bin% cat yasuoka （ぽこ）
#! /bin/nawk -f
# "yasuoka" Version 2.1
{
  t=$0;
  while(i=index(t,"yasuoka"))
    t=substr(t,1,i-1) "YASUOKA" substr(t,i+7);
```

```
    printf("%s\n",t);
  }
  ~/bin% ■
```

うまくいくかな？

```
~/bin% echo yasuoka | yasuoka （ぽこ）
YASUOKA
~/bin% ■
```

お、正解。 while は変わってないんですね。

root ： 実は if と同じで、ちょっと変わってるけどね。

while(式)
  命令;
    式が 0 でもヌルでもない間、命令を繰り返し実行する。
for(命令;式;命令)
  命令;
    最初の命令を実行した後、式が 0 でもヌルでもない間、直後の命令と式の後に書かれた命令とを繰り返す。
continue;
    ループの最後にジャンプする。ループを繰り返す条件が成立していれば、再度ループを繰り返す。
break;
    ループから飛び出す。

root ： それから、代入式は冪乗が増えた。

変数=式
    変数に式の値を代入する。代入された値を返す。
変数+=式
    変数に式の値を加える。加えた結果を値として返す。
変数-=式
    変数から式の値を引く。引いた結果を値として返す。
変数*=式
    変数を式の値倍する。結果を値として返す。
変数/=式
    変数を式の値で割る。割った結果を値として返す。
変数%=式
    変数を式の値で割った余りを変数に代入する。代入した値を返す。
変数^=式
    変数を式の値乗する。結果を値として返す。
変数++
    変数に 1 加える。加える前の変数の値を返す。
変数--
    変数を 1 減らす。減らす前の変数の値を返す。
++変数
    変数に 1 加える。加えた後の変数の値を返す。
--変数
    変数を 1 減らす。減らした後の変数の値を返す。

yasuoka ： すると結局 nawk では、三角関数と乱数と冪乗が増えただけですか？

root ： いやいや、実は他にもたくさん増えてる。でもまあ今日は時間がないから、それはまた次回にしようか。

yasuoka ： 今回は awk の復習みたいでしたね。

root ： まあ、そう言わずに。それじゃまた次回。