スカラー

「スカラー」とは？

・Perl が扱う最も単純な型
・１つの値を持つ
・その値は、時には数値、時には文字列（数値、文字列どちらでもある）★ここ重要

リテラル（＝ソースコード中での表記方法）

```
# 数値リテラル
$wk1 = 99;
$wk2 = -40;
$wk3 = 0377;       # ８進数の 377       （１０進で 255）
$wk4 = 0xff;       # １６進数の FF      （１０進で 255）
$wk5 = 0x11111111; #  ２進数の 11111111（１０進で 255）


# 浮動小数点リテラル
$wk1 = 1.25;
$wk2 = 7.25e45;  # 7.25 かける 10 の 45 乗


# 文字列リテラル
$wk1 = ''; # 空文字列
$name = 'taro';
print "The name is $name.\n";       #「The name is taro.」＋改行
print "The name is ${name}san.\n";  #「The name is tarosan.」＋改行
print 'The name is ${name}san.\n';  #「The name is ${fred}san.」＋改行
```

■逆スラッシュエスケープ

文字並び	意味
\n	改行
\r	復帰
\t	タブ
\007	８進数で ASCII コードを指定（007 ＝ベル）
\x7f	１６進数で ASCII コードを指定（7f ＝削除文字）
\\	逆スラッシュそのもの
\"	ダブルクォート
\'	シングルクォート
\l	次の１文字を小文字にする
\L	これ以降、\E までを小文字にする
\u	次の１文字を大文字にする
\U	これ以降、\E までを大文字にする

\Q	これ以降、\E までの非単語構成文字の前に逆スラッシュを挿入する
\E	\L、\U、\Q の効果を終了させる

演算子

数値演算子

```
$wk1 = 1 + 1;  # 数値の 2
$wk2 = 3 * 4;  # 数値の 12
$wk3 = 10 / 3; # 数値の 3.3333...
$wk3 = 10 % 4; # 数値の 1
$wk4 = 2**3;   # $wk4 = 2 * 2 * 2; と同じ
$wk1 += 3;     # $wk1 = $wk1 + 3; と同じ
$wk1 ++;       # $wk1 = $wk1 + 1; と同じ
```

文字列演算子

```
$wk1 = "taro" . "san";  # 文字列の "tarosan"
$wk2 = "taro" x 3;      # $wk2 = "taro" . "taro" . "taro"; と同じ
$wk3 = 5 x 2;           # $wk3 = "5" . "5"; と同じ
```

数字と文字列間の自動変換 (2.3.4)

★ここ重要
スカラーデータに対して使用する演算子から、自動的に数値と文字どちらとして評価されるか決まる。

逆に言えば・・・

・スカラーデータだけでは、数値か文字どちらで評価されるかは決まらない。
・スカラーデータはその場の状況に応じて数値になったり文字になったりする。

例１）

```
$wk = "Z" . 5 * 7;     # $wk4 = "Z" . 35; と同じで "Z35" となる
```

説明すると、

・「.」と「*」、優先順位が高いのは「*」なので、まずこれから評価される。
・「*」は数値を期待しているので、「数値として 5 かける 7」を行う。
・結果「"Z" . 35」となる。
・「.」は文字列を期待しているので、「文字列として Z に 35 をくっつける」
・結果「"Z35"」となる。

例２）
ソース

```
#!/usr/bin/perl -w
$wk = "";
$wk .= 1;   # "" . "1"
```

```
$wk += 2;   # 1 + 2
$wk .= 3;   # "3" . "3"
$wk += 4;   # 33 + 4
print "ans : ($wk) \n";
```

結果

```
ans : (37)
```

比較演算子

	数値	文字列
等しい	==	eq
等しくない	!=	ne
より小さい	<	lt
より大きい	>	gt
より小さいか等しい	<=	le
より大きいか等しい	>=	ge

ブール値

Perl には独立したブール型はない。
スカラー値を使用する。

偽（false）

・undef、0、''、'0'

真（true）

・上記以外の全て

ブール値の反転には、単項否定演算子「!」を使う。

未定義値

未初期化の変数の初期値は、特別な値「undef」を持つ。

・undef は数値でも文字列でもない。
・文字列として使うと、""（空文字列）として評価される。
・数値として使うと、0 として評価される。
・undef を判断するには、defined 関数を使う

```
if (defined($hoge)) {
  undef ではない場合
} else {
  undef の場合
}
```

リストと配列

「リスト」とは？

・スカラーの集合に順序をつけて並べたもの
・データそのもののこと

「配列」とは？

・リスト型の変数のこと

まとめ

・リストはデータ、配列は変数
・配列でないリストは存在するが、すべての配列は値としてリストをもつ

リストリテラル（＝ソースコード中での表記方法）

```
@wk1 = (1, 2, 3);     # ３つの値 1,2,3 からなるリスト
@wk2 = ("taro", 4.5); # ２つの値 "taro" と 4.5 からなるのリスト
@wk3 = ();            # 空リスト（要素がない）
@wk4 = (1 .. 10);     # 1 から 10 の整数リスト
@wk5 = ($a, $a+1);    # 要素中には、式も含めることができる
```

以下リストリテラルは全て同じこと

```
@wk1 = ('taro', 'jiro', 'saburo');
@wk2 = qw/ taro jiro saburo /;
@wk3 = qw{
  taro
  jiro
  saburo
}
```

リスト代入

リスト値を変数に代入できる。

```
($a, $b) = ("aaa", "bbb");
($a, $b) = ($b, $a);         # 入れ替えもできる
```

配列の要素にリストを代入

```
#!/usr/bin/perl -w
($wk[0], $wk[1]) = ("aaa", "bbb");
print "ans : " . join(",", @wk) . "\n";
```

```
ans : aaa,bbb
```

「@」は全ての配列要素を意味するので、以下のように書き換えることができる。

```
#!/usr/bin/perl -w
@wk = ("aaa", "bbb");
```

```
print "ans : " . join(",", @wk) . "\n";


ans : aaa,bbb
```

スカラー変数とリスト変数（＝配列）の名前空間は異なるので、同じ変数名を使っているように見えても実はそうではない。

```
#!/usr/bin/perl -w

$wk1 = "aa";
$wk1[0] = "AA[0]";
$wk1[1] = "AA[1]";

$wk2 = "bb";
@wk2 = ("BB[0]", "BB[1]");

print "ans1 : " . $wk1 . "\n";
print "ans2 : " . join(",", @wk1) . "\n";

print "ans3 : " . $wk2 . "\n";
print "ans4 : " . join(",", @wk2) . "\n";


ans1 : aa
ans2 : AA[0],AA[1]
ans3 : bb
ans4 : BB[0],BB[1]
```

リスト操作

■ pop、push 演算子

配列を、属に言うスタックに見立てた動作を行う。

・pop：配列の最後の１要素抜き出す
・push：配列の最後に１要素追加する

```
#!/usr/bin/perl -w

@nums = ('one', 'two', 'three', 'four', 'five');
print "nums:" . join(",", @nums) . "\n";

$ret = pop(@nums);
print "pop ret:$ret \n";
print "nums:" . join(",", @nums) . "\n";

$ret = push(@nums, 'six');
print "push ret:$ret \n";
print "nums:" . join(",", @nums) . "\n";

$ret = push(@nums, ('seven', 'eight'));
print "push ret:$ret \n";
print "nums:" . join(",", @nums) . "\n";


nums:one,two,three,four,five
pop ret:five
nums:one,two,three,four
push ret:5
nums:one,two,three,four,six
push ret:7
nums:one,two,three,four,six,seven,eight
```

■ shift、unshift 演算子

・shift：配列の最初の１要素抜き出す

・unshift：配列の最初に１要素追加する

```
#!/usr/bin/perl -w

@nums = ('one', 'two', 'three', 'four', 'five');
print "nums:" . join(",", @nums) . "\n";

$ret = shift(@nums);
print "shift ret:$ret \n";
print "nums:" . join(",", @nums) . "\n";

$ret = unshift(@nums, 'six');
print "unshift ret:$ret \n";
print "nums:" . join(",", @nums) . "\n";

$ret = unshift(@nums, ('seven', 'eight'));
print "unshift ret:$ret \n";
print "nums:" . join(",", @nums) . "\n";


nums:one,two,three,four,five
shift ret:one
nums:two,three,four,five
unshift ret:5
nums:six,two,three,four,five
unshift ret:7
nums:seven,eight,six,two,three,four,five
```

■ foreach 制御

値のリストの各要素に対して、ブロック中の処理を行う。
このとき、ブロック内で制御変数を書き換えるとリスト中のデータそのものも書き換わる。

```
#!/usr/bin/perl -w
@nums = ('one', 'two', 'three', 'four', 'five');
print "nums:" . join(",", @nums) . "\n";
foreach $num (@nums) {
  $num = '-' . $num . '-';
  print "$num \n";
}
print "nums:" . join(",", @nums) . "\n";


nums:one,two,three,four,five
-one-
-two-
-three-
-four-
-five-
nums:-one-,-two-,-three-,-four-,-five-
```

■ reverse、sort 演算子

・reverse：リストの要素の順序を逆にする
・sort：リストの要素をソートする

```
#!/usr/bin/perl -w
@nums = ('one', 'two', 'three', 'four', 'five');
print "nums:" . join(",", @nums) . "\n";

@nums_rev = reverse(@nums);
print "nums_rev:" . join(",", @nums_rev) . "\n";

@nums_srt = sort(@nums);
print "nums_srt:" . join(",", @nums_srt) . "\n";


nums:one,two,three,four,five
nums_rev:five,four,three,two,one
nums_srt:five,four,one,three,two
```

コンテキストの概念

式がどんな場所に置かれているかを示す概念で、Perl は式を解析する際に、スカラー値またはリスト値のどちらが期待されているかを判断する。

右辺 @nums は同じでも、コンテキストの違いで解釈がかわってくる。
scalar 関数を使うことで強制的にスカラーコンテキストを提供することも可能。（下の例では、print はリストコンテキストを要求するので scalar 関数を使うことで、強制的にスカラーコンテキストを提供している）

```
#!/usr/bin/perl -w
@nums = ('one', 'two', 'three', 'four', 'five');
@a = @nums;
$b = @nums;

print "nums:@nums\n";
print "a:@a\n";
print "b:$b\n";

print "scalar(a):".scalar(@a)."\n";

nums:one two three four five
a:one two three four five
b:5
scalar(a):5
```

警告

起動オプション "-w" # perl -w
もしくは、レキシカル警告

正規表現例

ドット (.) で区切られた各数値の桁数を４桁固定にそろえる

ソース

```
#!/usr/bin/perl -w
$wk = "1.12.123.1234";
$wk =~ s/([0-9]+)\.([0-9]+)\.([0-9]+)\.([0-9]+)/sprintf("%04d%04d%04d%04d",$1,$2,$3,$4)/e;
print "ans : $wk\n";
```

実行結果

```
ans : 0001001201231234
```