データ型（data types）

変数名

変数名として使用できるのは、

- 英字
- 数字
- アンダースコア（_）

です。ただし数字は 1 文字目には使用できません。これは C++などと同様です。

データ型（data types）の種類

Perl には、次の 3 つの組み込みデータ型があります。

- スカラ (scalar)
- リスト (list) または スカラの配列 (arrays of scalars)
- ハッシュ (hash) または スカラの連想配列 (associative arrays of scalars)

変数の参照

変数を参照するときは、型によって異なる記号で指定します。

- スカラ … 「$」
- スカラの配列 … 「@」
- スカラの連想配列 … 「%」

> それぞれの型は異なる名前空間を持ち、$foo と@foo は**同名であっても異なる変数**として扱われます。

スカラ

スカラ値の参照

記述例	意味
$s	スカラ s の値

リスト（スカラの配列）

リストの参照

記述例	意味
$s[3]	配列@s の 4 番目の要素
$#s	配列@s の最後の添字
@a	配列全体。つまり$a[0],$a[1],…,$a[n]
@a[3,4,5]	$a[3],$a[4],$a[5]と同じ（配列スライス）

配列の要素の数を取得

scalar()関数で、スカラの配列の要素の数を取得できます。scalar - perldoc.perl.org

```
my @s = (1,2,3,4,5);

print $#s;          # 4

print scalar( @s ); # 5
```

配列の要素を削除しても、その要素が未定義値となるだけで、配列の大きさは変わりません。

```
delete( $s[ 2 ] );

print $#s;          # 4

print scalar( @s ); # 5



print "@s"; # 1 2  4 5
```

しかし末尾の要素を削除すると、配列の大きさは 1 つだけ小さくなります。

```
delete( $s[ 4 ] ); # 末尾の要素を削除

print $#s;          # 3

print scalar( @s ); # 4



pop( @s );         # 末尾の要素を削除

print $#s;          # 2

print scalar( @s ); # 3
```

配列のすべての値の参照

```
foreach my $value ( @a ) {

    print "$value¥n";

}
```

すべての値を並べて出力するだけならば、そのまま

```
print @a; # abc
```

としても可能です。このときダブルクォートで囲んで、

```
print "@a"; # a b c
```

とすると、スペースで区切られて出力されます。任意の文字で区切るには join()を用います。

```
print join( ':', @a ); # a:b:c
```

重複した要素の削除

配列のすべての要素の名前をキーとして、ハッシュを生成します。ハッシュは重複したキーを許さないため、そのようにハッシュを生成した時点で重複した要素の名前はなくなっています。そしてそのハッシュから、すべてのキーの名前を要素として配列を生成すれば、重複した要素が削除された配列を作成できます。配列やリストにある重複した要素を削除するには? - perlfaq4 - データ操作 - perldoc.jp

```
my @a = (1,2,'a','b','b',2,3);



my %h = map{$_,0} @a;

# または、

# my %h;

# @h{ @a } = ();



my @newArray = keys( %h );

print "@newArray"; # 1 a 3 b 2
```

ハッシュ（スカラの連想配列）

ハッシュの参照

記述例	意味
$s{ 'foo' }	ハッシュ%s の'foo'の要素
$s{ foo }	
@a{'x','z'}	$a{'x'},$a{'z'}と同じ
%h	ハッシュ全体

ハッシュのすべての値の参照

```
foreach my $key ( keys( %h ) ) {

    print "$key=$h{$key}¥n";

}
```

または、

```
while( my( $key,$value ) = each %h ) {

    print "$key=$value¥n";

}
```

とします。ハッシュのキーと値、それぞれの一覧を参照するには、keys()、values()を呼び出します。

```
my %h = ( a=>1, b=>2 );



print keys( %h );   # ab

print values( %h ); # 12
```

単純に print で出力すると、すべてのキーと値が連結されて出力されます。

```
print %h; # a1b2
```

リテラル（literal）

数値リテラル

```
my $a = 10;
```

リストリテラル

かっこで囲んで指定します。

```
my @a = ( 1,2,'a' );

$a[ 1 ] = 10; # 1,10,'a'

my @b = ( 1,2,3,4,5 );

@b[ 1..3 ] = ( 'a','b','c' ); # 1,'a','b','c',5
```

リストリテラルをスカラ変数に代入すると、リストの末尾の値が代入されます。

```
my $a = ( 1,2,3 );

print $a; # 3
```

一方でリストリテラルをリストに代入すると、そのリストを構成する要素にそれぞれ代入されます。

```
my( $a,$b,$c ) = ( 1,2,3 );

print $a; # 1

print $b; # 2

print $c; # 3
```

ハッシュの初期化

ハッシュは、リストリテラルを用いて初期化します。

```
my %a = ( 1,2,3,4 );
```

このときリストリテラルの要素はキーと値のペアと解釈され、この場合は

- 1 … キー
- 2 … キー1 の値
- 3 … キー
- 4 … キー3 の値

とみなされます。これをより明確にするには、=>演算子を用いて、

```
my %a = ( 1=>2, 3=>4 );
```

と記述することもできます。これは可読性を上げるための記法であり、意味は変わりません。

文字列リテラル

文字列リテラルは、シングルクォート (') またはダブルクォート (") で囲みます。このときダブルクォートでは文字列中の変数が展開され、さらにエスケープ シーケンスが評価されます。これは Unix シェルのクォートの扱いに準ずるものです。また PHP の文字列リテラルとも同様です。

```
my $a = 10;



print 'ABC $a ¥tA'; # ABC $a ¥tA

print "ABC $a ¥tA"; # ABC 10  A
```

変数の展開を期待するときは、変数名の後に変数名になり得る文字を記述してはなりません。変数 a を記述するとき、

```
print "X$aX"; # X と出力
```

とすると、「Use of uninitialized value $aX in concatenation (.) or string」として、後続する文字まで変数名とみなされてしまいます。このような場合には、変数名を{}で囲みます。

```
print "X${a}X"; # X10X と出力
```

このとき変数名の前の記号までかっこで囲むと、かっこ自体も出力されてしまいます。

```
print "X{$a}X"; # X{10}X と出力
```

この記述は PHP では有効ですが、Perl では無効です。

文字列の長さの取得

文字列の長さは length()で取得できます。length - perldoc.perl.org

```
print length( 'abc' ); # 3 と出力
```

ただしマルチバイト文字では、対象の文字列の文字エンコーディングとスクリプトのそれが一致していないと、正しい結果を取得できません。

```
print length( 'あいう' ); # 9
```

このような場合には、対象の文字列を UTF8 に変換した上で長さを取得します。

```
use Encode;

print length( decode( 'utf-8', 'あいう' ) ); # 3 と出力
```

または utf8 プラグマでスクリプトでの UTF-8 を有効にし、そして対象の文字列も UTF-8 とする方法もあります。utf8 - perldoc.perl.org

```
use utf8;

print length( 'あいう' ); # 3
```

こうすることで、そのスクリプト上での文字列リテラルの UTF8 フラグ（UTF8 flag）がオンとなります。ただしこのように UTF-8 を有効にすると、文字列をそのまま出力すると「Wide character in print」と警告されるようになります。What if I don't encode? - perlunifaq - perldoc.perl.org

これには utf8 パッケージの関数で、明示的にエンコードしてから出力するようにして対処します。Utility functions - utf8 - perldoc.perl.org

```
use utf8;



my $str = 'あいう';

utf8::encode( $str );

print $str;
```

または encode()関数を用いて、

```
use utf8;

use Encode;



print encode( 'utf-8', 'あいう' );
```

とする方法もあります。Encode::Unicode - perldoc.perl.org

なお encoding プラグマを用いて、

```
use encoding 'utf8';
```

とする方法もありますが、Perl 5.18 以降これは非推奨とされているため、上記のように utf8 プラグマを用います。

宣言（declaration）

ブロック内のローカルな変数とするには、my で宣言します。

```
my $a = 10;
```

my で宣言された変数には、ブロック外からはアクセスできません。

```
if( 1 ) {
    $a = 10;
    my $b = 20;
}
print $a; # 10
print $b; # Use of uninitialized value $b
```

strict とした場合には、この my を省くとエラーとなります。

```
@a = (1,2); # Global symbol "@a" requires explicit package name
```

リファレンス（references）

変数やサブルーチンへのリファレンス (参照) を取得し、それを通して対象にアクセスできます。たとえばこれを利用すれば、ハッシュの値にハッシュを格納できます。

```
my $s = {
    a=>{ x=>1, y=>2 },
    b=>{}
    };
```

定数

```
use constant PI => 3.14159;
```

このように定義した定数を、

```
PI = 3;
```

のように変更しようとすると、「Can't modify constant item in scalar assignment」としてコンパイル エラーとなります。

複数の定数

複数の定数を同時に定義するには、次のようにします。

```
use constant {
    A => 'a',
    B => 'b',
};
```

定数を 1 つずつ定義する場合は

```
use constant X => 'a','b','c';

print X; # abc と出力
```

のように複数のリテラルを指定することが可能ですが、これを複数を同時に定義する場合も同様にすると、

```
use constant {

    Y => 'a','b','c'

};

print Y; # a と出力
```

のように異なる結果となります。