関数（サブルーチン）

引数

仮引数

```
sub Func {

    return 1;

}
```

Perl の関数では引数部を記述しません。それはすべて「@_」という配列に格納されます。

```
sub Func {

    print $_[ 0 ]; # 10 と出力

    print $_[ 1 ]; # 20 と出力

}



Func( 10, 20 );
```

もし仮引数に名前を与えたいならば、配列をリストへ代入する方法を用いて、

```
sub Func {

    my( $a, $b ) = @_;



    print $a; # 10 と出力

    print $b; # 20 と出力

}
```

とすると、一括して変数に代入できます。仮引数の数と型を制限したいならば、プロトタイプを用います

実引数

関数の実引数を囲むかっこは、省略可能です。

```
print("Hello");

print "Hello";
```

基本的な文法の概要 - perlintro - Perl の概要 - perldoc.jp

また、デフォルト引数を取る関数では実引数も省略可能で、

```
print;
```

と記述すると、

```
print $_;
```

の意味となります。

ただし混乱の原因となるため、実引数のかっこは省略すべきではありません。

参照渡し（Pass by Reference）

Perl では、すべての引数は**参照渡し**です。それはデータ型には無関係です。

```
sub Func {

    $_[ 0 ] *= 2; # 受け取った引数を直接変更している。

}



my $x = 10; # 元の値はスカラの 10

Func( $x );

print $x;   # 20 と出力
```

- Pass by Reference - perlsub - perldoc.perl.org
- 参照渡し - perlsub - Perl のサブルーチン - perldoc.jp

プロトタイプ（Prototypes）

宣言と異なる引数で定義されたときに、エラーとなるように制限を設けられます。

```
sub Func( $ ); # これが宣言

...

sub Func() {} # Prototype mismatch: sub main::Func ($) vs ()
```

たとえば引数の数を制約できます。

```
sub Func( $$ ) {}

Func( 1 );     # Not enough arguments for main::Func

Func( 1,2 );

Func( 1,2,3 ); # Too many arguments for main::Func
```

引数の型にも制約を設けられます。

```
sub Func( ¥$ ) {}

Func( 1 ); # Type of arg 1 to main::Func must be scalar (not constant item)


my $a;

Func( $a );
```

これ以外にもさまざまな指定法があります。

- バックスラッシュ … 実引数の最初の文字を指定
- セミコロン … 必須の引数と省略可能な引数を分割