# ActivePerl*ではじめる CGI アプリケーション

千葉商科大学付属高等学校 数学科 樽　正人

2005 年 1 月 5 日 初版　2008 年 1 月 5 日 第 2 刷版

## 1　Perl を学ぼう

### 1.1　準備編

#### 1.1.1　Perl の概要

IT用語辞典 e-Words より

Perl[a]は，Larry Wall 氏が開発したプログラミング言語。テキストの検索や抽出、レポート作成に向いた言語で、表記法は C 言語に似ている。インタプリタ型であるため、プログラムを作成したら、コンパイルなどの処理を行なうことなく、すぐに実行することができる。CGI の開発によく使われる。とにかく機能が豊富なことで知られる。当初は UNIX 上で利用されたが、現在では Windows を含む様々なプラットフォームに移植されている。

[a] Practical Extraction and Report Language の頭文字

**■インタプリタ型とコンパイル型**　プログラム言語には，Perl のように記述したソースを実行しながら同時に解釈をしていく インタプリタ型 と，C 言語のように，記述したソースを先に機械にわかる形に直してから実行する コンパイル型 の 2 つに分類されます。

**コンパイル型**　訂正のたびにコンパイルをしなおす手間がありますが，実行速度が向上します。
**インタプリタ型**　訂正後すぐに実行できる手軽さがありますが，実行速度は劣ります。

**■Perl に出来ること**　プログラム言語ですから，ほぼ出来ないことはありません。特にネットワークコンピュータを制御する UNIX に標準装備され，ネットワーク上のデータを加工したり，作成したりすることができます。そのため WEB 上に見られる掲示板や Chat などのプログラムを開発する言語として定番になっています。

#### 1.1.2　Perl の基本構文

エディタにプログラムを記述することからはじめますが，

- 最初の行にはプログラムの本体である Perl が置いてある場所[*1]を指定します。
  **#!** の記号からはじまります。
- 次の行から，実際にプログラムを記述することになります。1 行ごとに処理を 1 つずつ記述しますが，必ず行末にデリミタ[*2]として **;** をいれます。
- **#**は，プログラムとして実行させたくない部分の頭に付ける特殊な記号です。

* http://www.activestate.com/Products/ActivePerl/
[*1] path（パス）といいます。一般に UNIX 環境では，usr 内にある bin というフォルダの中に，perl という名前でおかれています。
[*2] 区切りという意味。

次に例を示します。これは画面に”Hello World” とだけ表示するプログラムです。

```
#!c:\perl\bin\perl.exe   # 最初にパスをこのように指定します
print "Hello World";   # 最後にデリミタの; を忘れません。
```

構文として守らなければいけないことは，これだけです。

### 1.1.3 Perl を学ぶための環境準備

1. UNIX から，Windows 用に移植された ActivePerl という Perl を使用します。これは，すでに学校のコンピュータにインストールされています。
2. 作成したプログラムを保存する場所を作ります。 H:ドライブに，perl-study という名前のフォルダを新規に作成します。
3. プログラムを記述するために使用するエディタとして xyzzy を選びます。
4. Perl は Windows のプログラムと違い，コンソール[*3]上で実行されます。Windows 用のコンソールとして，”コマンドプロンプト ” が用意されています。

   スタートメニュー → すべてのプログラム → アクセサリ → コンマンドプロンプト

### 1.1.4 作業の流れ

1. xyzzy を使って，Perl のプログラムソースを記述します。
2. “ファイル名.pl” という形式で，先程作成した `H:\perl-study` に保存します。

   例えば，hello.pl という感じになります。

3. コンソールを起動したら，`H:\perl` に移動[*4]します。次にソースファイルを指定して Perl を実行します。

   ```
   （ 例えば上記の例で，hello.pl を実行する場合）
   H:\>cd perl-study     # perl-study に移動
   H:\PERL-STUDY>perl hello.pl     # hello.pl を実行
   ```

4. コンソール画面に，実行結果が表示されれば成功です。

### 1.1.5 コンソールの基本操作

実行にはコンソールでの操作が必要になります。ここでよく使うのコマンドを紹介します。

| コマンド名 | コマンドの説明 |
|---|---|
| `dir␣[/p]` | 現在のフォルダにあるファイルの一覧を出力します。一画面ずつ表示したい場合は，p オプションを与えて実行します。 |
| `cd␣`**<フォルダA>** | フォルダAに移動します。 |
| `mkdir␣`**<フォルダA>** | 新規にフォルダAを作成します。 |
| `copy␣`**<ファイルA>**`␣`**<ファイルB>** | ファイルAをファイルBという名前でコピーします。 |
| `del␣`**<ファイルA>** | ファイルAを永遠に削除します。 |

表 1　よく使うコンソールコマンド一覧

**練　習**　実際にコンソールを起動して，上記のコマンドを実行してみましょう。

[*3] 直接キーボードからコマンドを打ち込んでいくために用意された原始的な画面

[*4] コンソールのコマンド cd を使用します。

## 1.2 基礎編

### 1.2.1 Hello World の世界

1. 2 ページの 1.1.4 節の手順で xyzzy を起動したら，以下の内容を入力します。

```
#!c:\perl\bin\perl.exe

print ''Hello World'';
```

2. H:\perl-study に，hello.pl という名前で保存します。
3. コンソールを起動し，H:\perl-study に移動したら， perl hello.pl と実行します。
4. コンソールに Hello World と出力されたら成功です。

**練習** 上記のソースを改変して，

```
Hello World
Good Bye
```

と 2 行のメッセージがコンソールに出力するようにしましょう。

ヒント 改行命令は, \n

### 1.2.2 スカラー変数

Perl のすばらしさは，変数を自由に扱える点です[*5]。変数とは，数学の文字で言うと $x$ にあたります。いわばデータを格納する箱です。

スカラー変数には 1 度に 1 つのデータしか入りません。2 つも 3 つも入れる箱にはなれません。[*6]

スカラー変数は，スカラー変数の証として先頭に$をつけます。また，変数名として使える文字は，半角英ではじまる半角英数字と記号_（アンダーバー）だけです。数字からはじまる名前は使えません。

| よい例 $abc や $var1 | 悪い例 $123 や $1var |
|---|---|

スカラー変数を使ったプログラムの例を下に示します。

```
1 行目  #!c:\perl\bin\perl.exe                    ↓↓実行結果↓↓
2 行目  $a = '3';
3 行目  $b = '5';
4 行目  $ans = ' 答え';
5 行目   print '' スカラー変数のサンプル 1 \n\n'';
6 行目   print ''$a + $b の$ans は，'';
7 行目   print $a + $b;
```

実際に入力して実行する前に，右の余白に予測結果を書いてみよう。
予測したら，”var1.pl” という名前で保存し，実行結果を確認してみよう。
2,3,4 行目は変数$a と$b に数値，変数$ans に文字列を代入しています。5,6,7 行目で画面に結果を出力しています。
特に，6 行目の” で囲まれた + は単なる記号と解釈され，7 行目の + は演算子として計算されている点に注意しましょう。

---

[*5] Perl に限らずプログラム言語はどれも変数を扱えますが，プリミティブな言語ほどその扱いは厳密になります。変数の大きさを間違えるとバッファオーバーフローといってメモリからデータがあふれでる重篤なエラーを引き起こす原因になるからです。

[*6] 複数扱いたい場合には，配列変数を使用します。後述します。

**■シングルクオート（'）とダブルクオート（"）の違い**　Perl では，文字列を扱う場合には，必ず シングルクオート か ダブルクオート でその文字列を囲ま（くくるという）なければなりません。もし，その囲まれた文字列の中にスカラー変数（以降，変数という）が含まれていたとき，次の注意が必要です。[*7]

- シングルクオートでくくった場合には変数が展開されません。

**例）**

```
#!c:\perl\bin\perl.exe
$a = '3';
$b = '5';
print '$a と $b は展開されません';
```

**（結果）**

```
$a と $b は展開されません
```

- ダブルクオートでくくった場合には変数が展開されます。

**例）**

```
#!c:\perl\bin\perl.exe
$a = '3';
$b = '5';
print ''$a と $b は展開されます'';
```

**（結果）**

```
 3 と 5 は展開されます
```

**練習**　先程の var1.pl ソースを使用して，実際に確認してみましょう。

### 1.2.3　演算子の種類

演算子には，**２項演算子**と**単項演算子**の２種類があります。

**■２項演算子**　２つの値（引数という）を演算する記号[*8]のことです。次のようなものが用意されています。

| 演算子 | 解説 |
|---|---|
| ＋ | 加法（例　3+2 → 5） |
| - | 減法（例　3-2 → 1） |
| * | 乗法（例　3*2 → 6） |
| / | 除法（例　6/3 → 2） |
| ** | べき乗（例　3**2 → 3 の 2 乗） |
| ％ | 剰余（例　12％5 → 12 を 5 で割った余り） |
| . | 文字列連結（例　” あいう”.” えお” → あいうえお） |

**■単項演算子**　１つの値（引数）を対象に演算し，その結果を元の値に代入する演算記号です。

| 演算子 | 解説 |
|---|---|
| ++ | 引数となる変数に 1 を加え，その結果をその引数に代入する（例　$i++ や ++$i） |
| −− | 引数となる変数から 1 を引き，その結果をその引数に代入する（例　$i−− や −−$i） |

単項演算子が，前（++$i）にある場合は，その場で 1 を加えた値が代入されるので，

**（例）**

```
$i = 0;
print ++$i;
```

**（結果）**

```
1
```

となります。

[*7] var1.pl の 7 行目のように演算処理のみを行なう場合に，クオートでくくると演算記号は文字列として扱われます。

[*8] 例）3 + 2 など

それに対して，後ろ（$i++）にある場合は，代入が遅れますので，

**（例）** **（結果）**

```
$i = 0;                          0
print $i++;
```

となります。

単項演算子ならではの特徴です。注意しましょう。

**■演習問題**

1. $name1 という変数に**自分の苗字**を，$name2 という変数に**自分の名前**を代入し，**文字列連結演算子を利用して**，フルネームを画面に出力してみよう。

```
#!c:\perl\bin\perl.exe
$name1 = '山田';
$name2 = '太郎';

print （                    ）;
```

2. 次のカッコ内に適する数式を入れ，プログラムを完成させよう。

```
#!c:\perl\bin\perl.exe
$a = '2';
$n = '16';

# 画面に 2 の 16 乗の結果を出力する。
print ''2 の 16 乗の答えは，''.（              ）.''\n'';
```

3. 単項演算子（++） を利用して，次のカッコに適する数式を入れよう。

```
#!c:\perl\bin\perl.exe
$year = '2008';

# 画面に「今年は 2008 年だから，来年は 2009 年ですね。」と出力させる。
print '' 今年は，$year 年だから，来年は（          ）年ですね。\n'';
```

### 1.2.4 配列変数

1.2.2 節のスカラー変数には，一度に 1 つの値しか代入できませんでした。
それに対し，配列変数では一度に複数の変数を扱うことができます。
いわば，スカラー変数を 1 列に複数並べたイメージになります。

配列変数

スカラー変数 1，スカラー変数 2，スカラー変数 3，...

配列変数は，配列変数の証として先頭に**@マーク**をつけます。
**例）**`@array` **や** `@pairs`

**■配列変数への代入** 配列変数に値を代入するには，次の 2 つの方法があります。後者の qw を使えば文字列を' で囲む必要がなく，また，それぞれの区切りにスペースを用いる点に注意してください。

```
@array = ('apple','orange','melon');
@array = qw(apple orange melon);
```

**■配列変数から値の参照** 配列変数のそれぞれの変数は，添え字と呼ばれる 0 から始まる整数番号で識別されます。
上記の例の場合，apple の値が代入されている変数の添え字番号は 0 になります。
ですから，値を代入する方法として

```
$array[0]*9 = 'apple';
$array[1] = 'orange';
$array[2] = 'melon';
```

というのもあります。
また，逆に代入した値を参照したい場合は

| 例） | （結果） |
|---|---|
| `print $array[0];` | apple |

となります。

**■練習** 次のカッコ内に適するものを入れよ。

```
#!c:\perl\bin\perl.exe

# 配列変数@weeks に，'日','月',...,'土' を代入する。
(                                    );

# 今日の曜日を画面に出力させる。
print '' 今日は，(          ) 曜日です。\n'';
```

*9 先頭が@でなく，スカラー変数と同じ$である点に注意。

### 1.2.5 連想配列変数（ハッシュ変数）

（キーワードで）連想（できる）配列変数という意味になります。配列変数では，添え字を頼りにデータを取り出しましたが，連想配列変数では，添え字の変わりにキーワードを利用できます。
連想配列変数は，その証として先頭に **% マーク**をつけます。
例）%hash　や　%form

**■連想配列変数（ハッシュ変数）への代入**　代入方法は３つありまが，いずれもキーワードとデータの対（ペア）で代入します。

その１）
%hash = ('name' , 'Jane' , 'country' , 'America' , 'sex' , 'female');

その２）
%hash = ('name' => 'Jane' , 'country' => 'America' , 'sex' => 'female');
(=>) は (,) と同じです。ペアであることを視覚的に持たせるために用意されました。

その３）
$hash{'name'} = 'Jane';
$hash{'country'} = 'America';
$hash{'sex'} = 'female';

どちらも name というキーで Jane という値を，country というキーで America という値を，sex というキーで female という値を連想配列変数（ハッシュ変数）%hash に代入したことになります。

**■連想配列変数（ハッシュ変数）から値の参照**　配列変数と違い，欲しい値はキーとして定義した言葉から容易に参照できます。

例）
ハッシュ変数 %hash から Jane という値を取り出すには，$hash{'name'}[*10]とします。

```
%hash = ('name','Jane','country','America','sex','female');
print "$hash{'name'}\n";
```

上の場合，画面には Jane と出力されます。

**■練習**　ハッシュ変数 %foods について，

（１）meat をキーで cow という値，drink をキーで milk という値，soup をキーで corn という値を代入し，
（２）milk という値を画面に出力する

プログラムを hash_ex.pl という名前で作成しなさい。

[*10] 先頭が % でなく，スカラー変数と同じ$である点に注意。

## 1.3 実践編

### 1.3.1 条件分岐 if と else 文

条件を事前に用意しておき，その条件を満たす時と満たさない時で処理内容を変更するときに if 文を使用します。文法は次のようになります。

```
if(条件){
  条件を満たす時の処理
}
else{
  条件を満たさない時の処理
}
```

例）変数$num の値が 1 ならば成功と画面に出力。それ以外なら失敗と出力。

```
$num = 1 # ここの値を色々変えてみる。
if($num == 1){
  print "成功";
}
else{
  print "失敗";
}
```

ここで，条件の中に”==”という記号がありますが，このように左辺値と右辺値を比較する演算子のことを**比較演算子**といいます。

**■主な比較演算子の紹介**　if 文を使うときに，比較演算子は必要になります。ここで主な比較演算子を紹介します。

| 比較演算子 | 解説 |
|---|---|
| == | 数値 1 == 数値 2 というように使います。等しければ true です。 |
| != | 数値 1 != 数値 2 というように使います。等しくないとき true です。 |
| eq | 文字列 1 eq 文字列 2 というように使います。等しければ true です。 |
| ne | 文字列 1 ne 文字列 2 というように使います。等しくないとき true です。 |
| > | 数値 1 > 数値 2 というように使います。数値 1 の方が大きいとき true です。 |
| >= | 数値 1 >= 数値 2 というように使います。数値 1 と等しいか大きいとき true です。 |
| < | 数値 1 < 数値 2 というように使います。数値 1 との方が小さいとき true です。 |
| <= | 数値 1 <= 数値 2 というように使います。数値 1 と等しいか小さいとき true です。 |

### 1.3.2 ファイルの入出力処理　open 関数

open 関数を使えばデータをファイルに出力したり，逆にファイルからデータを入力したりすることができます。文法は次のようになります。

ファイルから入力するとき，

```
open(ファイルハンドル名[*11],"<入力（取り込む）ファイル名");
```

例）ABC.txt ファイルのデータを入力して，画面に出力。

```
open(IN,"<ABC.txt");
print <IN>;        # ファイルハンドルを直接扱うときは<>で囲む
close IN;          # 開いたファイルは，close 関数で必ず閉じること。
```

ファイルに出力するときは上書きと追書きの 2 つのモードがあり，

```
open(ファイルハンドル名,">出力（上書き）ファイル名");
open(ファイルハンドル名,">>出力（追書き）ファイル名");
```

例 1）XYZ.txt ファイルにデータを出力（上書きモード）。

```
open(OUT,">XYZ.txt");
print OUT "書き込みテスト\n";
close OUT;
```

例 2）XYZ.txt ファイルにデータを追書き出力（追書きモード）。

```
open(OUT,">>XYZ.txt");
print OUT "もう一度書き込みテスト\n";
print OUT "ここで終了。\n";
close OUT;
```

### 1.3.3 繰り返し処理　foreach 文

複数のデータを 1 つずつ取り出しながら，同じ処理を繰り返したい。そんな時に便利なのが，**foreach 文**です。**foreach 文**は，配列変数からデータがなくなるまで，一つずつ取り出します。文法は次のようになります。

```
foreach スカラー変数（配列変数）{
      処　理;
}
#配列変数から１つずつ順に取り出したデータを指定したスカラー変数に代入します。
#代入先のスカラー変数が省力された場合は，特殊変数$_に代入されます。
```

例 1）配列変数@fruits からデータを一つずつ取り出し，画面に出力。

```
@fruits = ('apple','orange','melon','grape','strawberry');
 foreach $value (@fruits){
     print "$value\n";
}
```

[*11] ファイルハンドルとは，ファイルを処理するためのオブジェクトです。一般に，**適当な半角英語の大文字**などで表します。

例 2）XYZ.txt ファイルから，データを一つずつ取り出して画面に出力。

```
# XYZ.txt からデータを入力して，配列変数@pairs に格納
 open(IN,"<XYZ.txt");
@pairs = <IN>;
 close IN;

# 配列変数@pairs からデータを１つずつ取り出す
 foreach $value (@pairs){
   print "$value\n";
}
```

**■練習問題**　次の&で区切られた データ の内容を data.txt というファイルに出力（書き出）してみよう。

データ

顧客コード&会社名&業種&従業員数&資本金
00001&大川製薬&製薬&2,000 人&2 億円
00002&川種物産&商事&15,000 人&5 億円
00003&ミヅノスポーツ&スポーツ用品&3,500 人&1 億 1 千万円
00004&マックテリア&飲食業&45,000 人&8 千万円
00005&MTT&通信事業&120,000 人&3 億 5 千万円

### 1.3.4　データの分割処理　split 関数

split 関数は，指定したコード（文字）でデータを切り分けたいときに使用します。切り分けたデータは，新たに配列変数に格納したりします。文法は次のようになります。

```
配列変数 = split(/区切り文字/, データ)
#指定した区切り文字でデータを切り分けて，配列変数に格納します。
```

例）データを区切り文字"&"で切り分けて，配列変数 @parts に格納したら画面に結果を 1 行ずつ出力。

```
$line = '幼年期&若年期&成年期&中年期&老年期';
@growth = split(/&/,$line);
print "犬は，５つの発達段階で成長します。\n";
print "生後間もない時期を，$growth[0] と言います。\n";
print "そして$growth[4] を迎え，個体差はありますが 15〜18 年でその生涯を閉じます。\n";
```

**■練習問題１**　例を参考にし，以下の 3 つの作業を実行するプログラム部分をそれぞれ記述しよう。

例）$lines = 'abc0def0ghi0jkl' を 0 で切り分けて，配列変数@parts に入れる。
答）@parts = split(/0/,$lines);

1. 前節 1.3.4 で作成した data.txt を open 関数でいったん配列変数@values に読み込む。
   答）

2. 前問の配列変数@values で顧客コード 00003 の行にあたる$values[3] を split 関数を使って&で切り分けて，新たな配列変数@parts に入れる。
   答）

3. 前問の配列変数@parts から会社名だけを取り出して，画面に出力する。
   答）

**■練習問題２**　次のスカラー変数$buf から=の右辺値だけを取り出し，一行ずつ画面に出力するプログラム ex-split.pl を作成しよう。

```
$buf = 'var1=apple&var2=orange&var3=melon&var4=grape&var5=strawberry';
```

手順

1. &を区切り文字として split して適当な配列変数に格納する。
2. 次に格納した配列変数から foreach 文でデータを一つずつ取り出しながら，=でまた split しながら，画面に出力する

### 1.3.5 エラーと例外処理

プログラム（コンピュータ上に限らず）に，エラーはつきものです。プログラム自体が招くエラーと周りの環境から招かれるエラーとがあります。前者の場合は，そのプログラム自体の問題ですから, その部分を改変することでエラーを無くせます。しかし，後者の場合は利用するユーザーによって環境もさまざまであるため，エラーを確実に無くすことは不可能と言えます。

予測できない環境によるエラーでは，正常にプログラムが動作しないまでも大切なデータだけは保護する処理が必要になります。**こういった処理を例外処理といいます**。

特に前節 1.3.2 で利用した open 関数の場合，プログラムを実行する環境によっては指定したファイルを作成できなかったり，読み込みに失敗することなどがあります。open 関数は，処理に成功した場合には true の 1 を，失敗した場合には false の 0 を返します。**これを戻り値といいます**。

open 関数に限らず，perl で使われる関数では一般に成功した場合は True の 1 を，失敗した場合には False の 0 を戻り値として返すようになっています。
この戻り値を使えば，次のような if 文を使った 例外処理 が可能です。

```
#!c:\perl\bin\perl.exe

if(open(OUT,">>XYZ.txt") == 0){
      print "Error\n";
      exit;
}
print <IN>;
close IN;
```

**■練習問題**　前節 1.3.4 の例 2）に例外処理を加えたプログラム foreach-samp-err.pl を作成しよう。

### 1.3.6 サブルーチン化

次は ABC.txt の内容を XYZ.txt にコピーするものです。同じような例外処理を 2 回記述し，冗長です。

冗長な例外処理

```
#!c:\perl\bin\perl.exe
# 入力モードで ABC.txt を開くが，開けないときは Error と画面に出力して終了する
if(open(IN,"<ABC.txt") == 0){
      print "Error";
      exit;
}
@lines = <IN>; # ABC.txt の内容を配列変数@lines に読み込む
close IN; # ファイルを閉じる

# 出力モードで XYZ.txt を開くが，同じように開けないときは Error と画面に出力して終了する
if(open(OUT,">XYZ.txt") == 0){
      print "Error";
      exit;
}
print OUT @lines; # @lines の内容をファイルハンドル OUT を通して XYZ.txt に出力
close OUT; # ファイルを閉じる
```

下は，例外処理の部分を関数（サブルーチン）化したものです。作成した関数は，**&関数名**で呼び出します。

関数化した例外処理

```
#!c:\perl\bin\perl.exe
# 入力モードで ABC.txt を開くが，開けないときは例外処理関数&Error を呼び出す
if(open(IN,"<ABC.txt") == 0){
     &Error;
}
@lines = <IN>; # ABC.txt の内容を配列変数@lines に読み込む
close IN; # ファイルを閉じる

# 次に出力モードで XYZ.txt を開くが，同じように開けないときは例外処理関数&Error を呼び出す
if(open(OUT,">XYZ.txt") == 0){
     &Error;
}
print OUT @lines; # @lines の内容をファイルハンドル OUT を通して出力する
close OUT; # ファイルを閉じる

# 関数 Error の記述
sub Error
{
      print "ファイルエラー";
      exit;
}
```

関数（サブルーチン）化するときは，頭に sub[*12]を付けます。文法は次のとおりです。

[*12] sub は，subroutine（お決まりの仕事）の頭文字 3 つ

```
sub 関数名
{
    処理
}
```

**■関数の引数**　関数を実行する際に，メッセージを添えることができます。引数を渡すと言います。引数は，特殊な配列変数@_に格納されて関数に渡されます。
例）前述の｛関数化した例外処理｝に引数を持たせます。

引数を利用する例外処理関数

```
#!c:\perl\bin\perl.exe
# 入力モードで ABC.txt を開くが，開けないときは引数を付けて\&Error を呼び出す
if(open(IN,"<ABC.txt") == 0){
     &Error("ファイル入力");
}
@lines = <IN>; # ABC.txt の内容を配列変数@lines に読み込む
close IN; # ファイルを閉じる

# 出力モードで XYZ.txt を開くが，開けないときは引数を付けて\&Error を呼び出す
if(open(OUT,">XYZ.txt") == 0){
     &Error("ファイル出力");
}
print OUT @lines; # @lines の内容をファイルハンドル OUT を通して出力する
close OUT; # ファイルを閉じる

# 関数 Error の記述
sub Error
{
    $msg = $_[0]; ## 今回は引数が１つなので，配列変数@_の最初のデータを$msg に代入
     print "$msg エラー";
     exit;
}
```

上の例のように引数の内容を変えることで，例外がどこの処理で発生したかを調査することができます。
また，引数はカンマで区切れば複数渡すこともできます。

**例）**

```
#!c:\perl\bin\perl.exe

print "長方形の面積を求める。\n";
&rectArea(4,5);

sub rectArea # ２つの引数を下にして長方形の面積を求める関数
{
  $tate = $_[0];
  $yoko = $_[1];
  $kotae = $tate * $yoko;
   print "答え：$kotae";
}
```

# 2 CGI アプリケーションの作成

## 2.1 環境整備編

### 2.1.1 CGI アプリケーションの保存先

### 2.1.2 CGI アプリケーションの属性

## 2.2 基礎編

### 2.2.1 HTML の出力

### 2.2.2 フォームからデータの送信

## 2.3 実践編

### 2.3.1 フォームからデータの受信

### 2.3.2 データの処理

### 2.3.3 ファイルの出力

### 2.3.4 セキュリティの向上