Perl によるプログラミング入門 in 東海

第二回 - プログラムの構造化

2016/03/27 第一版

目 次

---

1 はじめに

1.1 お久しぶり

お久しぶりです．前回のテキストを読んでもらったのかもしれません．読んでもらっていないかもしれません．このテキストは，前回のテキストを読んだ人向けに書いています．題材が同じなので，もし前回のテキストをまだ読んでいなければ，参照しながらだと読みやすいと思います．

前回は，プログラムを勉強し始めた人向けの内容を目指しました．今回は，プログラムは一通り書けるけど，その次の段階の人向けです．やっつけ仕事より，もう少し長く使うプログラムを書くことを想定しています．まったくの初心者ではないけど，まだなんとなく書いている人，プログラミングの基礎を一緒に学びましょう．

プログラミングの世界へようこそ．サポータはあなたを手助けをします．このテキストは，一人でも読めるように考えて書きますが，必要であれば，誰かに助けを求めてください．ここに居るならサポータが．一人で読んでいるなら，インターネットの誰かが助けてくれるでしょう．

---

2 おさらい

2.1 TODO リスト

前回は TODO リストを作りました． 作った TODO リストいくつか機能がありました．一覧にします．

- TODO リストを保存するファイルを初期化できる
- TODO リストを追加する
- TODO リストの TODO を全部表示する
- TODO リストの TODO を完了にする
- TODO リストのうち未完了な TODO だけを表示する

2.2 できるようになったこと

プログラミングのやり方をいくつか学びました.

- 準備
  - プログラムファイルをテキストファイルで編集する
  - プログラムファイルをコンソールで実行する
- Perl
  - 意図
    - `#` から行末までは，コメントとなり，読む人に意味を伝える
    - 空行は，意味のかたまりを分割するために入れる
  - 単一の値，スカラ
    - 文字列，数値，`undef`，ファイルハンドル がある
    - 真偽値 (ブーリアン) としても使われる
      - 偽: `""`, `"0"`, `0`, `undef`
      - 真: それ以外
    - スカラ変数は，`$` が先頭
  - 複数のスカラ値を持つ，リスト
    - スカラを複数をひとまとめにする
    - リスト変数は，`@` が先頭
    - `[]` と添字 (数字) で個別のスカラ値を参照できる.
    - スカラ変数のように扱うには，`$` + 変数名 + `[` + 添字 + `]`
    - 添字は，`0` から始まる
  - 変数
    - 値に別名をつける機能
    - 代入 `=`(イコール) によって，変数と値をひもづける
    - 最初に使うときに `my` をつける
  - 演算子
    - 1 つ以上のスカラ値をスカラ値，又はリスト値に変換する
    - スカラでも特にブーリアンを入力，出力に持つ演算子もある
    - 引数の位置で呼び出し方は色々
      - ブーリアン → ブーリアン: `not` + 引数
      - 数値，数値 → 数値: 引数 1 + `+` + 引数 2

        - ファイルハンドル → 文字列 or 文字列のリスト: < + 引数 + >
- 関数
    * 0 個以上の引数与えて，何か処理をさせる．返り値がある．
    * 関数名 + (+ 引数 1 + , + 引数 2 + … +)
    * `print()`
        - 引数をファイルハンドルに出力する．ファイルハンドルを指定しなければコンソールに出力する
    * `open()` / `close()`
        - ファイルにひもづけられたファイルハンドルを作ったり，破棄したりする
    * `chomp()`
        - 文字列の最後の改行文字を削除する
        - 値を返すというよりは，指定された変数の文字列そのものを変更する
    * `push()`
        - 指定されたリスト変数の末尾に要素を追加する
    * `split()`
        - 文字列を分解したリストを返す
- 制御構造
    * `if` で，処理するかしないかを制御できる
        - `if (A) {B}` の形式で使う．A はブーリアン．B は 0 個以上の式.
    * `while` で処理を繰り返すことができる
        - `while (A) {B}` の形式で使う．A はブーリアン．B は 0 個以上の式.
        - A の結果が変わらないと繰り返しが終わらないので注意
    * `foreach` でも処理を繰り返すことができる
        - `foreach` + 繰り返し変数 + (+ 繰り返すリスト値+) + `{B}` の形式で使う
        - リストが最初にあるので，繰り返す回数はリスト値の個数
- コンテキスト
    * 何が求められているかで，演算子や関数の戻り値が変わる．その求められ方
    * スカラコンテキスト
        - スカラ値を期待されているコンテキスト
        - 例: スカラ変数への代入 (=)，演算子の引数，`if` などの条件部分，
    * リストコンテキスト

- リスト値を期待されているコンテキスト
- 例: リスト変数への代入 (=)，関数の引数.

3　確認するプログラムを書く

前回のテキストをやった人は，結果を目で確認してきたと思います．プログラムを便利に使う次の一歩として，自分の書いたプログラムの動作を自分の書いたプログラムで確認できるようにしましょう．

前回のテキストをやっていない人は，`01_hello.pl` をコピーして，実行できるようにしておいてください．

現在のディレクトリに `01_hello.pl` が存在していて，`./01_hello.pl` の形式で実行可能なことを確認してから次のステップに進んでください．

又，これ以降も前回のテキストで作ったファイルを使いますので，事前に前回使用したファイルを全部作成しておくのも良いでしょう．

3.1　シンプルな例

3.1.1　`01_hello.t`

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More; # is() / done_testing() 関数を使うために必要

# プログラム名の指定
my $program = "./01_hello.pl";

# プログラムの実行，実行結果の取得
my $got = `$program`;

# 実行結果 (got)，と 期待する結果 (expected) の比較
my $expected = "Hello, World\n";
```

```
16 my $message = "test for stdout";
17 is($got, $expected, $message);
18
19 # テスト実行完了を宣言
20 done_testing();
```

3.2 概略説明

.t 拡張子 (ファイル名の最後の . (ドット) 以降の文字列) は， テスト (test) の頭文字です．動作確認のことをテストと呼ぶことが多いようです． 拡張子は，.pl ではないですが，これも Perl 言語のプログラムファイルです．

の行，use strict; use warnings; の行は説明の必要はないでしょう．必要がある人は，前回のテキストを読んでみてください．

use Test::More; は，テストに使う関数を使えるようにしています．このプログラムでは，is() と done_testing() という関数を使っています．

is($got, $expected, $message) は, $got の中身が$expected と等しかったら確認 ok としています．$message は，結果を出力するときに，複数テストしたときにどのテストが ok かをわかりやすくするためのメッセージです．

done_testing() は，テストの終わりを意味します．

3.3 実行

細かい説明の前に，実行してみましょう． このファイルは Perl 言語のプログラムファイルですので， perl + プログラムファイルの形式で実行します．これも前回にやりました．

結果は，

```
1 ok 1 - test for stdout
2 1..1
```

のようになったと思います．

ok は，確認できた．という意味です．is() 関数の出力と思うとよいでしょう． 全部が ok なので，すべてが問題ない状態です．

1..1 の部分は，結果の個数を数えています．今回は，全部で 1 回なので，このように表示されています．この部分はあまり気にしなくともよいです．

例えば，15 行目をこのように書き換えて実行してみましょう．

01_hello_ng.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More; # is() / done_testing() 関数を使うために必要

# プログラム名の指定
my $program = "./01_hello.pl";

# プログラムの実行，実行結果の取得
my $got = `$program`;

# 実行結果 (got)，と 期待する結果 (expected) の比較
my $expected = "aaa" . "Hello, World\n";
is($got, $expected, "test for stdout");

# テスト実行完了を宣言
done_testing();
```

結果は，not ok を表示して，何やらメッセージが出るようになりました．

```
not ok 1 - test for stdout
#   Failed test 'test for stdout'
#   at 01_hello_ng.t line 16.
#          got: 'Hello, World
# '
#     expected: 'aaaHello, World
# '
1..1
# Looks like you failed 1 test of 1.
```

これは，実行したプログラムの結果として"Hello, World\n" が得られた (get) けど，期待していた (expected) のは "aaaHello, World\n" だったよ．ということを意味します．結果と期待したものの両方を表示してくれています．

このように，プログラムでプログラムの結果を確認することができます．

これくらい簡単なプログラムでは，自分の目で確認するよりも， `ok` / `not ok` を表示させる方が，むしろ面倒ですが，結果が複雑だったり，確認する項目や回数が増えてくると，テストプログラムで確認した方が楽になります．

3.4 解説

では，使っている機能の説明をしましょう．

3.4.1 モジュール / プラグマ

プログラムのパーツを封じ込めたものです． `perl` に最初から同梱されているものと， 後でインストールするものがあります．自分で作ることもできます． `use` でその機能を使う宣言をします．

機能はそれぞれのモジュール毎に異なります．モジュール名は大抵大文字から始まります (例: `Test::More`)．`strict`, `warnings` は，プラグマ (pragma) といってモジュールとは異なりますが， `use` を使う点，プログラムに機能を追加するという点では同じものです．

3.4.2 `Test::More` モジュール

`Test::More` テストを書くための機能を提供するモジュールです．

今回主に使うものは以下の関数です．

`is()` 関数 A is B のように，2 つのものが等しいことを確認するために使います．

1 番目の引数が，実行結果，

2 番目の引数が，期待した値です．

3 番目の引数が，メッセージです．テスト結果に表示できます．省略可能です．

`done_testing()` 関数 テストプログラムファイルの終わりに書きます．プログラムが確認したテストの個数を数えるのに使います．

3.4.3 `` (バッククォート) 演算子

`` バッククォート演算子は，` + 文字列 + ` の形式で， 文字列部分を実行した結果を返します．結果というのは，プログラムがコンソールに出力した部分です．print() 関数で出力していたものです．

今回は，自分が書いた Perl プログラムを指定していますが，システムに存在するプログラムを実行することもできます．

- 文字列 → 文字列 又は 文字列のリスト：` + 文字列 + `
  - 実行してコンソールに出力される内容が受け取れる

3.5 練習問題

1. システムに乗っているプログラムの実行結果を比較するテストプログラムを書いてください．(Windows なら dir，MacOSX，Linux の人は ls などが比較的使い易いでしょう)
2. いくつかの演算子の機能を試してみてください (+, -, *, /, ., x, ==, eq, not, and, or, …)
3. くりかえしの機能を試してみてください (while, foreach の中や外で is を使ってみてください．ブロックの中で計算した結果を確認するのもよいでしょう)

---

4 プログラムの動作を確認するプログラム

前回のテキストで作った以下のプログラムの動作を確認するテストプログラムを書きます．

- 03_init_todo.pl
- 04_add_todo.pl
- 05_list_todo.pl
- 06_done_todo.pl
- 07_list_notyet_todo.pl

まず，例を示しましょう．

4.1 TODO リスト初期化プログラムをテストする

4.1.1 03_init_todo.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More; # is() / done_testing() 関数を使うために必要
use File::Temp ("tempdir"); # tempdir() 関数を使うために必要
use Cwd; # cwd() 関数を使うために必要

# 実行するファイル名を指定する
my $program_filename = "03_init_todo.pl";
my $cwd = cwd();
my $program_fullpath = join("/", $cwd, $program_filename);

# todolist.txt を出力するための一時ディレクトリを作成し，実行し，結果を比較する
# (この一時ディレクトリは自動的に削除される)
{
    # 一時ディレクトリの作成
    my $tmp_dirname = tempdir(CLEANUP => 1);

    # プログラムの実行
    chdir($tmp_dirname) or die; # 一時ディレクトリに移動する
    `$program_fullpath`; # プログラムの実行

    # 実行結果の取り出し
    open(my $fh, "<", "todolist.txt") or die;
    my $got = join("", <$fh>);
    close($fh) or die;

    # 実際の値 (got) と期待する結果 (expected) を比較
    my $expected = "sample todo\n"; # 期待する結果
    is($got, $expected);
}

done_testing(); # テストの終了を宣言
```

4.2 概略説明

急にプログラムが長くなりました． 長いので，コメントを入れておきました．前回にも書きましたが，空行で意味の区切りを作っています．

1. いつもの
2. 必要な関数の準備 → モジュール
3. 実行するファイル名の準備
4. 一時ディレクトリの準備
5. プログラムの実行
6. 実行結果の取り出し (データファイル)
7. 実際の値，期待する値の比較

プログラムを実行して，結果を比較する部分は，シンプルな例と同じです．

ポイントは，ディレクトリを変更してからプログラムを実行している部分です．意識していなかったかもしれないですが，前回書いたプログラム (`03_init_todo.pl`) は，データファイル (`todolist.txt`) を， 実行を開始したときと同じディレクトリに置きます．

実行する前に，ディレクトリを変えないとどうなるでしょうか？

前回テストを実行したときのデータファイルが残っていたり，最悪，自分の使っているデータファイルが消えてしまいます．これは嬉しくないでしょう．

そこで，このテストプログラムでは，テスト対象のプログラムを実行する前に，ディレクトリを移動しています．移動する先は，一時的に作ったもので，最初は空の状態です．テストプログラムが終了したらそのディレクトリは自動的に削除されます．

4.3 解説

4.3.1 `File::Temp` モジュール

一時的なファイルやディレクトリを作るための機能が含まれています．

`use File::Temp ("tempdir")` という書き方は，`File::Temp` の中の `"tempdir"` を使えるようにしています．

これは，`tempdir()` 関数を，`tempdir()` として呼ぶための書き方です．これが無くとも，モジュール名 + :: + 関数名 () の形式，

File::Temp::tempdir() で関数を呼び出すことができます．長いので，短く呼べるようにしました．

今までのモジュール (例えば，Test::More) では， use することで，is() 関数や done_testing() 関数が自動的に 関数名 () の形式で呼び出せるようになっていました．

File::Temp モジュールの File::Temp::tempdir() 関数のように， 自動では 関数名 () の形式で呼び出せずに，モジュール名::関数名 () の形式で呼び出すためには， use 時に引数を与える必要があるモジュールもあります．

引数として何を与えることができるかはモジュールによって異なります．詳しくは，各モジュールのドキュメントを参照してください．

tempdir() 関数 一時ディレクトリを作成して，ディレクトリ名を返します．

引数の CLEANUP => 1 は，作成した一時ディレクトリを後で自動的に削除することを関数に伝えています．

=> 演算子は，, (カンマ) 演算子とほぼ同じですが， 右側を" (ダブルクォート) なしで文字列として扱えるようにします．CLEANUP => 1 は，"CLEANUP", 1 と同じです． => は，少し短かく書けるのと，"CLEANUP" と 1 の 2 つの値の間に関係があることを示したいときに使います．

この場合の 1 は，真偽値の真の意味で，CLEANUP => 1 で CLEANUP が真である．という意図を表現しています．

4.3.2 chdir() 関数

現在のディレクトリを変更します．

プログラムが終了すれば，現在のディレクトリは，元に戻ります．

4.3.3 Cwd モジュール

cwd() 関数 現在のディレクトリ名を文字列で返します．

例えば，

10_directory_test.pl

```perl
#!/usr/bin/env perl

use strict;
use warnings;

use File::Temp ("tempdir");
use Cwd;

print("current directory: ", cwd(), "\n");

my $temp_dir = tempdir(CLEANUP => 1);
chdir($temp_dir) or die;

print("new     directory: ", cwd(), "\n");
```

出力例

```
$ perl 10_directory_test.pl
current directory: /Users/user/Documents/20160327
new     directory: /private/var/tmp/6tyZNeBm85
$ pwd         # Windows だと cd
/Users/user/Documents/20160327
```

ですと，一時ディレクトリに移動して，そのディレクトリを確認することができます．

4.3.4 ブロック

{ と } に囲まれた部分は，ブロックと呼ばれます．

前回のテキストの while, if の説明でも出てきました．

ブロックは，while では，その部分だけ繰り返し実行されたり，if では，その部分だけ実行される，されない，の対象になっています．ブロックは，複数のプログラムの命令文を 1 まとめにします．

読み易さのために，ブロックの中は，ブロックの外よりも先頭に空白文字を入れる字下げ を入れるようにします． 字下げは，エディタの機能のままでも良いですが，そのような機能が無いエディタを使っている場合には，1 つ分の字下げがスペースを 4 つと決めて編集すると良いでしょう．ブロックが，

1 つのかたまりを構成することを示すためです．(空行も同じように意味のかたまりを表現していました．)

```
{
    命令文 1; # 字下げ (空白 x4) されている
    命令文 2;
      :
}
```

ブロックの中にブロックがあれば，その内側のブロックは，2 つ分の字下げを入れます．これは，while の内側の if などで使ってきました．

```
{
    {
        命令文 1; # 2 階層分字下げ (空白 x8) されている
        命令文 2;
           :
    }
}
```

ブロックは，while や if などが無い部分にも使うことができます．そこまで意味はありませんが，字下げが入るので，1 つのまとまりを表現するのに便利です．

意味がないというのは嘘です． ブロックには，1 つ重要な機能があります．それは，変数の有効期限を指定することです．

```
#!/usr/bin/env perl

use strict;
use warnings;

{
    my $localvar = 1;
}
print($localvar);
```

これは，エラーになります．$localvar 変数の有効期限が my で宣言したブロックの内部だからです．

```
#!/usr/bin/env perl

use strict;
```

```
use warnings;

my $localvar;
{
    $localvar = 1;
}
print($localvar);
```

こうするとエラーなく実行することができます.

変数の有効期限のことを スコープ と呼びます.

変数への影響範囲が把握しやすくなるので，スコープをできるだけ小さくすると プログラムが読み易くなります.

4.4 練習問題

1. File::Temp::tempdir() という形式で 10_directory_test.pl を書き直してみましょう．同じ動作になりましたか？
2. 10_directory_test.pl を描き直して，10_directory_test.t を作りましょう．'is() 関数を使って，chdir() の動作を確認しましょう.

   (ヒント: MacOSX の場合，chdir() 関数で指定したディレクトリ名と cwd() 関数から得られたディレクトリ名が異なっており，先頭に"/private" が付いていることがあります．比較側に"/private" を付けておくとよいでしょう．)

```
my $expected = "/private" . $temp_dirname;
```

3. ブロックを使って，変数の有効範囲を確かめてみましょう．プログラムのサンプルを書きます.

```
#!/usr/bin/env perl

use strict;
use warnings;

# pattern1
my $v1;
{
    $v1 = "assign in block";
```

```
    print($v1);
}
print($v1);

# pattern2
{
    my $v2 = "declarate in block\n";
    print($v2);
}
print($v2);

# pattern3
{
    print($v3);
    my $v3 = "use before declaration";
}
```

5　プログラムの共通部分をまとめる

5.1　TODO リストを追加するプログラムをテストする

この調子で，TODO を追加するプログラムをテストするプログラムを書いてみましょう．

5.1.1　04_add_todo.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;
use File::Temp ("tempdir");
use Cwd;

my $program_filename = "04_add_todo.pl";
```

```
11 my $cwd = cwd();
12 my $program_fullpath = join("/", $cwd, $program_filename);
13
14 {
15     my $tmp_dirname = tempdir(CLEANUP => 1);
16     chdir($tmp_dirname) or die;
17
18     my $new_todo_content = "append new todo";
19     open(my $wfh, "|-", $program_fullpath) or die;
20     print($wfh $new_todo_content, "\n");
21     close($wfh) or die;
22
23     open(my $rfh, "<", "todolist.txt") or die;
24     my $got = join("", <$rfh>);
25     close($rfh) or die;
26
27     my $expected = $new_todo_content . "\n";
28     is($got, $expected);
29 }
30
31 done_testing();
```

5.2 解説

5.2.1 open() の入力モード

open() 関数のモードが，"|-" になっています．これは，パイプと呼ばれて，そのファイルハンドルへの出力が，別のコマンドへの入力に繋がっている状態を作ります．

```
・元プログラム
    ファイルハンドルへの出力 (print)
    ↓↓↓
・別コマンドへの入力
    ファイルハンドルから入力 (<>)
```

逆に，`` (バッククォート) 演算子は，別のコマンドの出力をプログラムで受け取ることができました．これもパイプを使って書くこともできます．

open() 関数を使うと長くなるので， 本テキストでは `` を優先して使っています．

5.3 問題提起

03_list_todo.t と 04_add_todo.t を見比べてください． 同じ部分が多い気がしませんか？

1. ファイル名から，フルパス名 (現在のディレクトリを追加したファイル名) を得る部分
2. 一時ディレクトリに移動する部分
3. データファイルの内容を取り出して，期待通りか確認する部分

プログラムを共通化して使う方法について学びましょう．

5.4 プログラムの共通部分 - fullpath()

5.4.1 処理を別ファイルに移動する

まずは，03_init_todo.t を見てみます．

現在のディレクトリを付与したファイル名を得る部分から考えましょう．

この節では，長くなるのでコードはポイントだけ書くことにします．

```
my $program_filename = "03_init_todo.pl";
my $cwd = cwd();
my $program_fullpath = join("/", $cwd, $program_filename);
```

まず，どの部分がポイントかわかりやすいように，一旦ブロックの中に閉じ込めてみます (この時点では，実行できません)．

```
# 実行するファイル名を指定する
{
    my $program_filename = "03_init_todo.pl";
    my $cwd = cwd();
    my $program_fullpath = join("/", $cwd, $program_filename);
}
```

これだと，$program_fullpath 変数が後で使えないので，my の変数宣言だけ外に出します．

```
# 実行するファイル名を指定する
my $program_fullpath;
{
    $program_filename = "03_init_todo.pl";
    my $cwd = cwd();
    $program_fullpath = join("/", $cwd, $program_filename);
}
```

ブロックに名前を付けて，それを呼び出す形式にします． sub の部分は，ブロックに名前を付けています．名前を付けられたブロックは関数 です．return 文は，関数の返り値を作ります．関数の書き方，sub，return などについては後でも説明します．

関数を定義しているものを先に書くように順序を入れ替えました．関数は使う前に定義されていなければいけないからです．

```
# 実行するファイル名を指定する
sub fullpath { # 関数化
    my $program_filename = "03_init_todo.pl";
    my $cwd = cwd();
    return join("/", $cwd, $program_filename);
}

my $program_fullpath = fullpath();
```

このままでは，ファイル名の部分が固定になってしまいます．04_add_todo.t でも使うことを考えているので，ファイル名が変更できるように，関数の外側に出します．関数はファイル名を受け取れるような形式に書き換えます．

```
# 実行するファイル名を指定する
sub fullpath {
    my ($program_filename) = @_; # 引数を受け取る
    my $cwd = cwd();
    return join("/", $cwd, $program_filename);
}
my $program_filename = "03_init_todo.pl";
my $program_fullpath = fullpath($program_filename);
```

04_add_todo.t でも使えるようにするため，03_init_todo.t とも別のファイルを作り，03_init_todo.t からそこに関数を移動します．

このときに，Cwd モジュールの use も一緒に移動します．何故ならば，

fullpath() 関数の中で cwd() 関数を使っているからです.

共通化するためのファイルには書き方がありますので，これも関数と同様に後で説明します.

とりあえず以下のように書けば，use MyTestUtil できるようになります.

MyTestUtil.pm

```
package MyTestUtil;

use strict;
use warnings;

use Cwd;

use Exporter ("import");
our @EXPORT_OK = ("fullpath");

sub fullpath {
    my ($program_filename) = @_;
    my $cwd = cwd();
    return join("/", $cwd, $program_filename);
}

1;
```

03_add_todo.pl は，関数を移動されたので，スッキリしました．use Cwd も削除します.

```
# use Cwd を削除
use MyTestUtil ("fullpath"); # use MyTestUtil を追加
```

```
# 実行するファイル名を指定する
my $program_filename = "03_init_todo.pl";
my $program_fullpath = fullpath($program_filename);
```

また，04_add_todo.t も同じような編集をして fullpath() 関数を使うようにします.

できあがったプログラム全体を載せます.

5.4.2 03_init_todo.t - fullpath 適用後

```perl
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More; # is() / done_testing() 関数を使うために必要
use File::Temp ("tempdir"); # tempdir() 関数を使うために必要

use MyTestUtil ("fullpath");

# 実行するファイル名を指定する
my $program_filename = "03_init_todo.pl";
my $program_fullpath = fullpath($program_filename);

# todolist.txt を出力するための一時ディレクトリを作成し，実行し，結果を比較する
# (この一時ディレクトリは自動的に削除される)
{
    # 一時ディレクトリの作成
    my $tmp_dirname = tempdir(CLEANUP => 1);

    # プログラムの実行
    chdir($tmp_dirname) or die; # 現在のディレクトリを一時ディレクトリにする
    `$program_fullpath`; # プログラムの実行

    # 実行結果の取り出し
    open(my $fh, "<", "todolist.txt") or die;
    my $got = join("", <$fh>);
    close($fh) or die;

    # 実際の値 (got) と期待する結果 (expected) を比較
    my $expected = "sample todo\n"; # 期待する結果
    is($got, $expected);
}

done_testing(); # テストの終了を宣言
```

5.4.3 04_add_todo.t - fullpath 適用後

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;
use File::Temp ("tempdir");

use MyTestUtil ("fullpath");

my $program_filename = "04_add_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    my $tmp_dirname = tempdir(CLEANUP => 1);
    chdir($tmp_dirname) or die;

    my $new_todo_content = "append new todo";
    open(my $wfh, "|-", $program_fullpath) or die;
    print($wfh $new_todo_content, "\n");
    close($wfh) or die;

    open(my $rfh, "<", "todolist.txt") or die;
    my $got = join("", <$rfh>);
    close($rfh) or die;

    my $expected = $new_todo_content . "\n";
    is($got, $expected);
}

done_testing();
```

5.4.4 MyTestUtil.pm

```
package MyTestUtil;

```

```
3  use strict;
4  use warnings;
5
6  use Cwd;
7
8  use Exporter ("import");
9  our @EXPORT_OK = ("fullpath");
10
11 sub fullpath {
12     my ($program_filename) = @_;
13     my $cwd = cwd();
14     return join("/", $cwd, $program_filename);
15 }
16
17 1;
```

そこまで短くなっていないですが，以上が複数ファイルにまたがるような，処理の共通化，関数化の例です．

処理を共通化することで，処理が短くなったり，同じ処理は同じ名前 (関数名) で呼び出せるようになります．

関数化しておくと，自分の意図していた複数の処理のかたまり (ブロック) に名前を付けることができ，実際にその名前で呼び出すことができます．又，その名前が表すものが，思っていたことと違っていたら，関数の中身を編集することで，それを呼び出している部分全部に反映されます．

5.5 解説

5.5.1 関数の書き方

関数は用意されているもの (`open()`, `print()` など) もありますが， 自分で書くこともできます．

変数が値に名前を付けたものだったように，関数はブロックに名前を付けたものです．別の場所でその名前でそのブロックを使うことができます．ブロックは複数の命令を束ねているので，関数名はその命令群の意味や意図を表す名前を付けます．

`sub` キーワードを使って，`sub` + 関数名 + ブロックの形式で， 関数を定義できます．

関数を呼び出したときに引数は， 関数の定義する場合は，@_ 変数に入ると考えることができます．

func(1, 2, 3) のように呼び出せば， 関数が動作するときには，@_ が (1, 2, 3) になっています．

関数の戻り値は，return で指定します．

関数は，return を実行すると，残りの処理を実行せずに，呼び出し元に戻ります．

サンプル

```
# 関数定義
sub times_join {
    my ($d, $str, $n) = @_;

    my @strs;
    my $i = 0;
    while ($i < $n) {
        push(@strs, $str);
        $i = $i + 1;
    }

    return join($d, @strs);
}

# 関数呼び出し
print(times_join("-", "abc", 3)); # => "abc-abc-abc"
```

my ($arg1, $arg2, ...) = @_; のように引数を受け取る書き方は， よく使うので覚えておきましょう．

又，関数は呼び出す前に定義されていなければ呼び出せません．モジュールの中に書けば，自然とそうなるので，モジュールの中に関数を書いているうちは，あまり気にすることはありません．

5.5.2 モジュールの書き方

いくつか知っておくべきことがあります．

Perl では，別ファイルの機能を使うのに，モジュールという仕組みを使いま

す．Test::More, File::Temp などは，Perl が標準で用意しているモジュールです．

自分でモジュールを書くこともできます．

ファイル名は，モジュール名 + .pm でなければなりません．MyTestUtil というモジュール名なら，ファイル名は MyTestUtil.pm です．

モジュールファイルの先頭で package + モジュール名の形式で，宣言をします．

モジュールファイルの最後には，1; を書きます．

モジュールは use されたら，モジュールファイルが実行されますが，成功したら真を返すことになっていて，エラーになった場合は，偽を返すことになっています．それが最後に，1; を書く理由です．

モジュールの書き方 (MyTestUtil モジュール)

```
1 package MyTestUtil;
2
3 sub fullpath {
4   ...
5 }
6
7 1;
```

使い方

```
1 use MyTestUtil;
2
3 my $ret = MyTestUtil::fullpath($arg);
```

モジュールの中の関数は，モジュール名と :: を先頭に付けて呼び出せます．

今まで，モジュールの中の関数は，モジュール名を指定しないで呼び出していました．

5.5.3 Exporter モジュール

関数を モジュール名 + :: + 関数名 () ではなく，関数名 () の形式で呼び出せるようにするために使っています．

以下の形式を今後も使うので，覚えておくと良いでしょう．

```
package YourPackage;
# モジュール側
use Exporter ("import");
our @EXPORT_OK = ("func1");
```

```
# 呼び出し側
use YourPackage ("func1");

func1();
```

our というキーワードが出ています．これは my と同じように使います． 今回は特に説明をしません．

5.5.4 MyTestUtil モジュール (自作)

今回作ったモジュールは，テストプログラムを書くためのパーツを集めたものです．

今のところ 1 つの関数しかありません．

fullpath() 関数 1 つめの引数はプログラムファイル名で，現在のディレクトリと連結することで，絶対パス形式の文字列を返します．

> 絶対パス形式とは，ディレクトリやファイルの位置を表現する方法です．現在のディレクトリからの相対的な場所を示すのは，相対パス．現在のディレクトリに関係なく，トップのディレクトリから途中の経路を全部記述するのを絶対パスと呼びます．
>
> chdir() を使った場合などで，いつものディレクトリと違う場所から目的のプログラムを指定して呼び出すときなどに便利です．

5.6 プログラムの共通部分 - chtempdir()

5.6.1 処理を別ファイルに移動する

もう少し共通部分を増やしてみましょう．

まずは，03_init_todo.t を見てみます．

一時ディレクトリを作成して，そこに移動するまでを関数にしてみましょう．

この節でも，コードはポイントを押さえて書くことにします．

```
{
    my $tmp_dirname = tempdir(CLEANUP => 1);

    chdir($tmp_dirname) or die;
```

この 2 行が対象の部分です．

まずは，ブロックにしてみましょう．

```
{
    {
        my $tmp_dirname = tempdir(CLEANUP => 1);

        chdir($tmp_dirname) or die;
    }
```

既にブロックの中なので，字下げが 2 段階になっています．これを関数定義にします． 使うより前に関数定義するようにしましょう．

```
sub chtempdir {
    my $tmp_dirname = tempdir(CLEANUP => 1);
    chdir($tmp_dirname) or die;
}

{
    chtempdir();
```

これを MyTestUtil モジュールに移動します．

MyTestUtil.pm には，関数の移動の他に関数内部で使っているモジュールの移動， @EXPORT_OK の追加をします．

MyTestUtil.pm

```
use File::Temp ("tempdir");

(中略)

our @EXPORT_OK = ("fullpath", "chtempdir");

(中略)

```

```
sub chtempdir {
    my $tmp_dirname = tempdir(CLEANUP => 1);
    chdir($tmp_dirname) or die;
}
```

03_init_todo.t も書き換えます.

03_init_todo.t

```
# use File::Temp ("tempdir"); の削除
use MyTestUtil ("fullpath", "chtempdir"); # 関数を追加
```

```
{
    chtempdir();
```

これで，chtempdir() 関数の移動は完了です.

04_add_todo.t も同じような編集をして， chtempdir() 関数を使うようにします.

できあがったプログラム全体を載せます.

5.6.2 03_init_todo.t - chtempdir 適用後

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More; # is() / done_testing() 関数を使うために必要

use MyTestUtil ("fullpath", "chtempdir");

# 実行するファイル名を指定する
my $program_filename = "03_init_todo.pl";
my $program_fullpath = fullpath($program_filename);

# todolist.txt を出力するための一時ディレクトリを作成し，実行し，結果を比較する
# (この一時ディレクトリは自動的に削除される)
{
    # 一時ディレクトリの作成
```

```
    chtempdir();

    # プログラムの実行
    `$program_fullpath`; # プログラムの実行

    # 実行結果の取り出し
    open(my $fh, "<", "todolist.txt") or die;
    my $got = join("", <$fh>);
    close($fh) or die;

    # 実際の値 (got) と期待する結果 (expected) を比較
    my $expected = "sample todo\n"; # 期待する結果
    is($got, $expected);
}

done_testing(); # テストの終了を宣言
```

5.6.3 04_add_todo.t - chtempdir 適用後

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir");

my $program_filename = "04_add_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    chtempdir();

    my $new_todo_content = "append new todo";
    open(my $wfh, "|-", $program_fullpath) or die;
    print($wfh $new_todo_content, "\n");
    close($wfh) or die;
```

```
20
21      open(my $rfh, "<", "todolist.txt") or die;
22      my $got = join("", <$rfh>);
23      close($rfh) or die;
24
25      my $expected = $new_todo_content . "\n";
26      is($got, $expected);
27  }
28
29  done_testing();
```

5.6.4 MyTestUtil.pm - chtempdir 適用後

```
1   package MyTestUtil;
2
3   use strict;
4   use warnings;
5
6   use Cwd;
7   use File::Temp ("tempdir");
8
9   use Exporter ("import");
10  our @EXPORT_OK = ("fullpath", "chtempdir");
11
12  sub fullpath {
13      my ($program_filename) = @_;
14      my $pwd = cwd();
15      return join("/", $pwd, $program_filename);
16  }
17
18  sub chtempdir {
19      my $tmp_dirname = tempdir(CLEANUP => 1);
20      chdir($tmp_dirname) or die; # 現在のディレクトリを一時ディレクトリにする
21      return $tmp_dirname;
22  }
23
24  1;
```

5.7 練習問題

1. 円周を計算する関数 `circumference_of_circle()` (引数: 半径，戻り値: 円周) を書いてみましょう．

   半径を入力して，円周を出力するプログラムを書いてみましょう．

   (ヒント: 円の円周は，$2\pi r$ です．π は `3.14` として計算すると良いでしょう)

2. モジュール `Math` を作り，その中に円周を作る関数，円の面積を作る関数 `area_of_circle()` (引数: 半径，戻り値: 面積) を書いてみましょう．

   `Math` モジュールを使って，半径を入力して円周と面積を出力するプログラムを作ってみましょう．

   (ヒント: 円の面積は，πr^2 です)

3. `is_todolist_content()` という関数 (引数: $expected, $message，戻り値: なし) を `MyTestUtil` モジュール内に作り，それぞれのテストで使うようにしてみましょう．

6 共通部分を使ってプログラムを書いてみる

6.1 TODO リストを表示するプログラムをテストする

`05_list_todo.t` を書いてみましょう．

6.1.1 `05_add_todo.t`

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir");

```

```
10 my $program_filename = "05_list_todo.pl";
11 my $program_fullpath = fullpath($program_filename);
12
13 {
14     # 準備
15     chtempdir();
16
17     open(my $wfh, ">", "todolist.txt") or die;
18     print($wfh "some todos", "\n");
19     print($wfh "todo2", "\n");
20     close($wfh) or die;
21
22     # 実行
23     my $got = `$program_fullpath`;
24
25     # 結果比較
26     my $expected = join("",
27                         "1:some todos\n",
28                         "2:todo2\n",
29                        );
30     is($got, $expected);
31 }
32
33 done_testing();
```

6.2 解説

新しいことがないので，解説するところがありません．

6.3 練習問題

1. テストするときに，事前にデータファイルが存在する状態にするために，データファイルを作っています．データファイルを作成する `make_datafile()` 関数 (引数: 文字列`$content`，戻り値: なし) を `MyTestUtil` モジュールの中に作って，`05_list_todo.t` を書き換えましょう．

2. 06_done_todo.t, 07_list_notyet_todo.t を作りましょう．

3. それぞれのテストプログラムで共通化できる部分があったら，MyTestUtil モジュールに移動してみましょう．

   (ヒント: プログラムにデータを入力する関数，辺りが候補になるでしょう)

4. (難しい) テストプログラムは MyTestUtil モジュールを作って同じ機能を複数の実行可能プログラムから呼び出すようにすることができました．

   TODO リストプログラムの機能すべてを，MyTodolist モジュールの中に入れて，そこから呼び出すだけにしてみましょう．

   (ヒント: 例えば，1 つのプログラムファイルの主要部分を全部関数として MyTodolist モジュールに入れてしまいましょう．init_todo.pl だったら，MyTodolist::init_todo 関数を作ります．元のプログラムはその関数を呼び出すだけにするのです．それでテストプログラムを流してみて，全部テストが通るようなら同じ動きをするようになっています．)

   (ヒント: テストしようとすると，各コマンドから MyTodolist モジュールが見付からないというエラーが出ると思います．use MyTodolist より前に以下を入れてみてください．)

```
1 use FindBin;
2 use lib ($FindBin::Bin);
```

---

7 機能追加

TODO リストのいくつかのプログラムをテストすることができるようになりました．

- 03_init_todo.t
- 04_add_todo.t
- 05_list_todo.t

残りは，各自やってもらうとして．これで，プログラム側の中身を変えても同じように動いているかどうかの確認ができます．これらのプログラムのテストを行う準備ができたと言えます．

- 03_init_todo.pl
- 04_add_todo.pl
- 05_list_todo.pl

それでは，プログラムに機能の追加してみましょう．

7.1 TODO リストを絞り込む

05_list_todo.pl に機能を追加してみましょう．

文字列を与えて，それと部分マッチするものだけ表示するようにしましょう．

7.1.1 05_list_todo.pl - マッチ機能追加

```
#!/usr/bin/env perl

use strict;
use warnings;

my $re;
if (@ARGV > 0) {
    $re = $ARGV[0];
}

open(my $fh, "<", "todolist.txt") or die;
my $count = 1;
while (defined(my $line = <$fh>)) {
    if (not(defined($re)) or $line =~ m/$re/) {
        print($count, ":", $line);
    }
    $count = $count + 1;
}
close($fh) or die;
```

7.1.2 05_list_todo.t - マッチ機能追加

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir");

my $program_filename = "05_list_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    # 準備
    chtempdir();

    open(my $wfh, ">", "todolist.txt") or die;
    print($wfh "some todos", "\n");
    print($wfh "todo2", "\n");
    close($wfh) or die;

    # 実行
    my $got = `$program_fullpath`;

    # 結果比較
    my $expected = join("",
                        "1:some todos\n",
                        "2:todo2\n",
                       );
    is($got, $expected);
}

{
    # 準備
    chtempdir();

    open(my $wfh, ">", "todolist.txt") or die;
```

```
38      print($wfh "some todos", "\n");
39      print($wfh "todo2", "\n");
40      close($wfh) or die;
41
42      # 実行
43      my $got = `$program_fullpath some`;
44
45      # 結果比較
46      my $expected = join("",
47                          "1:some todos\n",
48                         );
49      is($got, $expected);
50  }
51
52  {
53      # 準備
54      chtempdir();
55
56      open(my $wfh, ">", "todolist.txt") or die;
57      print($wfh "some todos", "\n");
58      print($wfh "todo2", "\n");
59      close($wfh) or die;
60
61      # 実行
62      my $got = `$program_fullpath soma`;
63
64      # 結果比較
65      my $expected = join("",
66                          );
67      is($got, $expected);
68  }
69
70  {
71      # 準備
72      chtempdir();
73
74      open(my $wfh, ">", "todolist.txt") or die;
75      print($wfh "some todos", "\n");
76      print($wfh "todo2", "\n");
```

```
77      close($wfh) or die;
78
79      # 実行
80      my $got = `$program_fullpath tod`;
81
82      # 結果比較
83      my $expected = join("",
84                          "1:some todos\n",
85                          "2:todo2\n",
86                         );
87      is($got, $expected);
88  }
89
90  done_testing();
```

7.2 解説

7.2.1 条件の考え方

`not(defined($re) or $str =~ m/$re/` は条件としてはちょっと長いです．2 つの条件を含むような条件は，`and` を使うべきか `or` を使うべきか， 更に `not` が入ってきたら混乱してしまう人も居るかもしれません．

考え方のサンプルを示します．

複雑な条件には表や図に書くのもよいかもしれません．視覚的に把握する方が理解が早まるかもしれません．

表 1: print したいかそうでないか

	`not(defined($re))`: 偽	`not(defined($re))`: 真 (`$re` が設定されていない)
`$str =~ m/$re/`: 偽	x print	o print
`$str =~ m/$re/`: 真 (`$str` が`$re` と比較 OK)	o print	o print

表 2: A or B の結果表

A or B	A: 偽	A: 真
B: 偽	偽	真
B: 真	真	真

このように表に実際に書いてみると，or であることは理解しやすいかと思います．

7.2.2 コマンドライン引数

05_list_todo.pl は，元の動きを変えないようにしました．

コンソールからの入力を受け取ったり，受け取らなかったりといった複数の場合のことを考えたくなかったので，それ以外の入力方法として，コマンドライン引数 を使っています．

コマンドライン引数とは，コマンドを実行するときに，コマンド名の後に書く文字列のことです．プログラムはコマンドライン引数を使って，動作を変えたりすることがあります．

```
# "1", "10", "abc" の 3 つがコマンドライン引数
$ ./command 1 10 abc
```

Perl プログラムの場合，@ARGV にコマンドライン引数が自動的に入るようになっているので，操作は簡単です．

my ($arg1, $arg2) = @ARGV のように関数の引数のように受け取ることもできます．

7.2.3 m// マッチ

m// は，=~ 演算子とひとまとめで覚えてください．$str =~ m/正規表現/ のように使います．正規表現と書いた部分は，かなり特殊なので別の項で説明します．

$str =~ m/正規表現/ は，文字列 と 正規表現を引数として， ブーリアンを返す演算子と考えることができます．

```perl
my @strs = ("first", "second", "third", "fourth", "fifth", "sixth", "seventh");

foreach my $str (@strs) {
    if ($str =~ m/th/) {
        print($str, "\n");
    }
}
# =>
# third
# fourth
# fifth
# sixth
# seventh
```

この例では，th が正規表現です．

演算子の前の文字列を正規表現と比較した場合， この場合は，文字列が th を含んでいれば，真を返します．

7.2.4 正規表現

正規表現は，文字列の集合を表現するためのものです．1 つの正規表現が複数の文字列を意図します．

例えば，aaa|bbb という正規表現は，文字列"aaa" と "bbb" を意図しています．

Perl の正規表現の場合，$str =~ m// の形式で比較すると，通常は，文字列が，その正規表現 (の意図する文字列) を含んでいると真を返します．

```perl
$str =~ m/aaa|bbb/;
```

この例の場合，文字列 $str が "aaa" 又は"bbb" という文字列を 含んでいれば真になります．

この例の，| (パイプ) 文字のように，正規表現の中で，特別な意味を持つ文字以外は単純に文字を意図します．

正規表現のなかで特殊な意味を持つ記号をいくつか載せておきます．これらは，基本的には \ (バックスラッシュ) を直前につけることで， 意味を打ち消すことができます．

| (パイプ) 正規表現の，or です．

`[]` (文字クラス) 1 文字分の `or` です.

`[ab]` なら `"a"` 又は `"b"` を表します.

`[0-9]` のように，`-` (ハイフン) で範囲も表現できます． これは，`0 or 1 ‖ 9` の全てのどれかの 1 文字を表現しています.

`.` (ドット) 1 文字ならなんでも良いです.

例えば，`a..b` なら `"a"` の後 2 文字なんでもよい文字が続いて， その直後に `"b"` が続く文字列を意図していることになります.

`?` (クエスチョン) 回数指定．直前の表現の 0 回又は 1 回の繰り返しを意図します.

例えば，`https?` だったら，`"http"` と `"https"` を意図します.

`*` (アスタリスク) 回数指定．直前の表現の 0 回又は それ以上の繰り返しを意図します.

`a*b` だったら，`"b"` や `"ab"` や `"aaaab"` を意図しています.

`+` (プラス) 回数指定．直前の表現の 1 回又はそれ以上の繰り返しを意図します.

`a+b` だったら，`"ab"` や `"aaaab"` を意図しています.

`"b"` は，先頭の `"a"` の部分を 0 回繰り返しているので，意図したものではありません.

`()` (括弧) 表現をグルーピングします.

回数指定のところに出てきた，直前の表現のという部分に関係します.

`(abc)+` という表現は，`"abc"` や `"abcabc"` を意図します.

`a(1|2)z` の場合は，`"a1z"` 又は `"a2z"` を意図します．`or` の範囲を指定することもできます.

7.3 練習問題

1. 名前をコマンドラインから受け取って，`"Hello " +` 名前を出力するよう なプログラムを書きましょう.

   コマンドライン引数を指定しなければ，`"john"` が与えられたようにしてください.

2. コマンドラインで数値を 2 つ与えて，2 つの数値の間の整数を合計するプログラムを書いてください．

   引数の数値が足りなければ，エラーメッセージを表示して終了するようにしてください．

   `./ex7_2.pl 1 10` のように実行されたら `55` を表示しましょう．

   (ヒント: 小数を整数に変換するには `int()` 関数を使います．`int(1.5) #=> 1`)

   (ヒント: プログラムを即座に終了するには `exit()` 関数を使います．関数の `return` に似ています．)

3. `Ex7` というモジュールを作って，2 つの数値の間の整数を合計する `sum()` 関数 (引数: $b, $e，戻り値: 数値) を作って下さい．

   それを使うようにプログラムを書き換えてください．

4. 先程のプログラムで，数値に見えるもの以外が入力されたらメッセージを出して終了するようにしてください．

   (ヒント: 数値は数字 `[0-9]` と `.` (ドット)，`-`(マイナス) などで表現できます．)

   (ヒント: 1 つの入力をチェックする `is_number()` 関数 (入力: 文字列，出力: ブーリアン) を作っておくのもよいでしょう)

5. 書いたプログラムをテストするプログラムを書いてみましょう．期待通り動いたでしょうか？

6. `Ex7` モジュールをテストするプログラムを書いてみてください．

---

8 まとめ

TODO リストを操作するプログラムをテストするプログラムを書きました．テストプログラムを書く過程で，プログラムを分割して再利用する方法を学びました．それ以外では，正規表現についていくつか学びました．

- テストプログラム
- プログラムのモジュール化
- 正規表現

テストプログラムは書きましたが，練習問題にあるように，肝心の TODO リストプログラムのモジュール化が進んでいませんので，自分で考えてみるのもよいでしょう．

---

9 おさらい

この会で Perl プログラミングについて学んだことをおさらいします．

9.1 テストプログラム

テストプログラムを書くのに，Test::More モジュールの is() と done_testing() 関数を使います．

9.1.1 Test::More モジュール

is() 関数 テストするのに使います．

実際の値，期待する値，メッセージ (省略可) を引数に取ります．

done_testing() 関数 テストの終了を表すために使います．

9.1.2 File::Temp モジュール

tempdir() 関数 一時ディレクトリを作るのに使います．

CLEANUP => 1 を引数に取って，作ったディレクトリを自動的に削除します．

9.1.3 chdir() 関数

現在のディレクトリを変更します．

プログラムを終了すると現在のディレクトリは元に戻ります．

9.1.4 Cwd モジュール

cwd() 関数 現在のディレクトリを返します.

9.1.5 `` (バッククォート) 演算子

`` (バッククォート) 演算子 ` + 文字列 + ` で，文字列をコマンドとして実行したときのコンソールへの出力されるものを文字列として返します.

9.1.6 open() のパイプモード

open(my $fh, "|-", コマンド名) "|-" は，パイプモードです.

コマンド名を実行して，そのコマンドがコンソールから受け取る入力に渡すことができるファイルハンドルを作ります.

9.2 ブロック

{} (ブロック) 複数の処理をひとまとめにします.

my で宣言した変数のスコープを閉じ込めます.

9.3 関数

sub + 関数名 + ブロック 関数を宣言した後で，関数名 + () で呼び出せるようにします.

サンプルを書いておきます.

```
sub func {
    my ($arg1, $arg2) = @_;

    my $ret = $arg1 + $arg2;

    return $ret;
}
```

9.4 自作モジュール

サンプルを書いておきます.

```
package YourModule;

use Exporter ("import");
our @EXPORT_OK = ("func1");

sub func1 {
}

1;
```

短い名前で呼び出せるように，@EXPORT_OK で関数名を書くのを忘れないようにしてください.

使う側のサンプル

```
use YourModule ("func1");

func1();
```

9.4.1 Exporter モジュール

サンプルを書いておきます． モジュールを自分で宣言するときに使います.

```
use Exporter ("import");
our @EXPORT_OK = ("func1");
```

9.5 正規表現

9.5.1 m//

$str =~ m/正規表現/ の形式で，正規表現が文字列を表していれば真を，そうでなければ偽を返します.

9.5.2 正規表現

普通の文字 特殊な意味を持たない文字は，それ自身を表現しています．

. (ドット) なんでもよい 1 文字を表現します．

[] (文字クラス) 1 文字の選択肢を表現します．

文字クラスの表現の中では， 1 文字ずつ指定するので，.(ドット) も . そのものを表しますが，一部の文字だけが特殊な意味以外の意味を持ちます．

- (ハイフン) を使って範囲も指定できます．(ex. [0-9]) - 文字自体を表すには，文字クラスの先頭に書くか --- のように- 自身のみの範囲を作ります．

^ (ハット) は，文字クラスの先頭で反転を意味します．例えば，[^0-9] は，0 ‖ 9 以外の 1 文字を意味します．a などの数字でない文字 1 文字分を表現しています．

? (クエスチョン) 量指定．0 又は 1 回の繰り返しを表現します．

+ (プラス) 量指定．1 又は それ以上の繰り返しを表現します．

*** (アスタリスク)** 量指定．0 又は それ以上の繰り返しを表現します．

() (括弧) 表現をグループ化します．

量指定の直前に使って，対象を指定するのに使ったりします．

| (パイプ) 表現の or を表現します．

\A 位置指定．文字列の先頭を意味します．

\Aabc は，"xxxabc" を意味しませんが，"abcxxx" は意味します．

\z 位置指定．文字列の末尾を意味します．

abc\z は，"abcxxx" を意味しませんが，"xxxabc" は意味します．

\A と一緒に使って，文字列全体を表現したいときにも使います．\Aabc\z は，"abc" を意味します．"xabcx" のように "abc" を含んでいるものではありません．

10　練習問題の答

10.1　3 練習問題

10.1.1　1

ex3_1.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

is(`ls`, join("",
        "01_hello.pl\n",
        "01_hello.t\n",
        "01_hello_ng.t\n",
        "02_calc.pl\n",
        "03_init_todo.pl\n",
        "03_init_todo.t\n",
        "04_add_todo.pl\n",
        "04_add_todo.t\n",
        "05_list_todo.pl\n",
        "05_list_todo.t\n",
        "06_done_todo.pl\n",
        "06_done_todo.t\n",
        "07_list_notyet_todo.pl\n",
        "07_list_notyet_todo.t\n",
        "MyTestUtil.pm\n",
        "ex3_1.t\n",
        "todolist.txt\n",
    ));

done_testing();
```

コマンド出力は大抵最後に改行が入っています．そのことに注意してください．

10.1.2 2

演算子の動作を自分で確認することができます.

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

is(1 + 1, 2, "1 + 1 #=> 2");
is(5 - 2, 3, "5 - 2 #=> 3");
is(5 * 2, 10, "5 * 2 #=> 10");
is(5 / 2, 2.5, "5 / 2 #=> 2.5");
is("a" . "b", "ab", "a . b #=> ab");
is("df" x 3, "dfdfdf", "df x 3 #=> dfdfdf");
is(100 == 101, "", "100 == 101 #=> F");
is(100 == 100, 1, "100 == 100 #=> T");
is("a" eq "b", "", "a eq b #=> F");
is("a" eq "a", 1, "a eq a #=> T");
is(not(1 == 1), "", "not(1 == 1) #=> F");
is(not(1 != 1), 1, "not(1 != 1) #=> T");
is(1 == 1 and 1 != 1, "", "1 == 1 and 1 != 1 #=> F");
is(1 == 1 and 1 == 1, 1, "1 == 1 and 1 == 1 #=> T");
is(1 != 1 or 1 != 1, "", "1 != 1 or 1 != 1 #=> F");
is(1 != 1 or 1 == 1, 1, "1 != 1 or 1 == 1 #=> T");

done_testing();
```

1 点良くないコードが含まれています. 偽を "", 真を 1 と決め付けて確認している点です.

このような真偽値を確認するテストには, ok() という関数が個別に用意されています. こちらを使った方が良いでしょう.

又, 真偽値をテストするのなら, 真, 偽の両方になるパターンでテストした方が良いでしょう.

10.1.3 3

繰り返しブロックの中で is を使うのは難しいですね.

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;
use File::Temp ("tempdir");

my @data = ("aaa", "bbb", "ccc");

{ # setup
    my $temp_dirname = tempdir(CLEANUP => 1);
    chdir($temp_dirname) or die;

    open(my $wfh, ">", "datafile") or die;
    foreach my $datum (@data) {
        print($wfh $datum, "\n");
    }
    close($wfh) or die;
}

{ # test for readline
    open(my $rfh, "<", "datafile") or die;
    my $buf = "";
    foreach my $expected (@data) {
        my $got = <$rfh>;
        is($got, $expected. "\n");
        chomp($got);
        $buf = $buf . $got;
    }
    close($rfh) or die;

    is($buf, join("", @data));
}

done_testing();
```

10.2　4 練習問題

10.2.1　1

use File::Temp ("temp") の引数部分を削除すると良いでしょう.

```
#!/usr/bin/env perl

use strict;
use warnings;

use File::Temp;
use Cwd;

print("current directory: ", cwd(), "\n");

my $temp_dir = File::Temp::tempdir(CLEANUP => 1);
chdir($temp_dir) or die;

print("new     directory: ", cwd(), "\n");
```

10.2.2　2

```
#!/usr/bin/env perl

use strict;
use warnings;

use File::Temp;
use Cwd;
use Test::More;

my $temp_dir = File::Temp::tempdir(CLEANUP => 1);
chdir($temp_dir) or die;

is(cwd(), $temp_dir);
#is(cwd(), "/private" . $temp_dir); # MacOSX の場合
```

```
15
16 done_testing();
```

10.2.3 3

サンプルの通りです.

10.3 5 練習問題

10.3.1 1

入力方法を覚えていましたでしょうか？

関数は使う前に宣言する必要があります.

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5
6  sub circumference_of_circle {
7      my ($r) = @_;
8      return 2 * $r * 3.14;
9  }
10
11 my $r = <STDIN>;
12 print(circumference_of_circle($r), "\n");
```

10.3.2 2

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5
6  use Math ("circumference_of_circle", "area_of_circle");
```

```
7
8  my $r = <STDIN>;
9
10 print(circumference_of_circle($r), "\n");
11 print(area_of_circle($r), "\n");
```

```
1  package Math;
2
3  use strict;
4  use warnings;
5
6  use Exporter ("import");
7  our @EXPORT_OK = ("circumference_of_circle", "area_of_circle");
8
9  sub circumference_of_circle {
10     my ($r) = @_;
11     return 2 * 3.14 * $r;
12 }
13
14 sub area_of_circle {
15     my ($r) = @_;
16     return 3.14 * $r * $r;
17 }
18
19 1;
```

10.3.3 3

Test::More は，MyTestUtil モジュールとテストプログラムの両方に必要なことに気をつけてください．

03_init_todo.t

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5
6  use Test::More; # is() / done_testing() 関数を使うために必要
7
```

```
use MyTestUtil ("fullpath", "chtempdir", "is_todolist_content");

# 実行するファイル名を指定する
my $program_filename = "03_init_todo.pl";
my $program_fullpath = fullpath($program_filename);

# todolist.txt を出力するための一時ディレクトリを作成し，実行し，結果を比較する
# (この一時ディレクトリは自動的に削除される)
{
    # 一時ディレクトリの作成
    chtempdir();

    # プログラムの実行
    `$program_fullpath`; # プログラムの実行

    # 実行結果の取り出し
    # 実際の値 (got) と期待する結果 (expected) を比較
    my $expected = "sample todo\n"; # 期待する結果
    is_todolist_content($expected);
}

done_testing(); # テストの終了を宣言
```

04_add_todo.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir", "is_todolist_content");

my $program_filename = "04_add_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    chtempdir();

```

```
    my $new_todo_content = "append new todo";
    open(my $wfh, "|-", $program_fullpath) or die;
    print($wfh $new_todo_content, "\n");
    close($wfh) or die;

    my $expected = $new_todo_content . "\n";
    is_todolist_content($expected);
}

done_testing();
```

MyTestUtil.pm

```
package MyTestUtil;

use strict;
use warnings;

use Cwd;
use File::Temp ("tempdir");
use Test::More;

use Exporter ("import");
our @EXPORT_OK = ("fullpath", "chtempdir", "is_todolist_content");

sub fullpath {
    my ($program_filename) = @_;
    my $pwd = cwd();
    return join("/", $pwd, $program_filename);
}

sub chtempdir {
    my $temporary_directory = tempdir(CLEANUP => 1);
    chdir($temporary_directory) or die; # 現在のディレクトリを一時ディレクトリにする
}

sub is_todolist_content {
    my ($expected, $message) = @_;

    open(my $fh, "<", "todolist.txt") or die;
```

```
    my $got = join("", <$fh>);
    close($fh) or die;

    is($got, $expected, $message);
}

1;
```

10.4 6 練習問題

10.4.1 1

05_list_todo.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir", "make_datafile");

my $program_filename = "05_list_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    # 準備
    chtempdir();

    make_datafile(join("",
            "some todos", "\n",
            "todo2", "\n",
        ));

    # 実行
    my $got = `$program_fullpath`;

```

```
    # 結果比較
    my $expected = join("",
                        "1:some todos\n",
                        "2:todo2\n",
                       );
    is($got, $expected);
}

done_testing();
```

MyTestUtil.pm

```
package MyTestUtil;

use strict;
use warnings;

use Cwd;
use File::Temp ("tempdir");
use Test::More;

use Exporter ("import");
our @EXPORT_OK = ("fullpath", "chtempdir", "is_todolist_content", "make_datafile");

sub fullpath {
    my ($program_filename) = @_;
    my $pwd = cwd();
    return join("/", $pwd, $program_filename);
}

sub chtempdir {
    my $temporary_directory = tempdir(CLEANUP => 1);
    chdir($temporary_directory) or die; # 現在のディレクトリを一時ディレクトリにする
}

sub is_todolist_content {
    my ($expected, $message) = @_;

    open(my $fh, "<", "todolist.txt") or die;
    my $got = join("", <$fh>);
```

```
    close($fh) or die;

    is($got, $expected, $message);
}

sub make_datafile {
    my ($content) = @_;

    open(my $wfh, ">", "todolist.txt") or die;
    print($wfh $content);
    close($wfh) or die;
}

1;
```

10.4.2 2

06_done_todo.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir", "is_todolist_content", "make_datafile");

my $program_filename = "06_done_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    # 準備
    chtempdir();

    make_datafile(join("",
            "some todos\n",
            "some todos 2nd\n",
        ));
```

```
21
22      # 実行
23      my $target = 1;
24      open(my $pfh, "|-", $program_fullpath) or die;
25      print($pfh $target, "\n");
26      close($pfh) or die;
27      print("\n");
28
29      # 結果比較
30      my $expected = join("",
31                          "Done,some todos\n",
32                          "some todos 2nd\n",
33                         );
34      is_todolist_content($expected);
35  }
36
37  done_testing();
```

07_list_notyet_todo.t

```
1   #!/usr/bin/env perl
2
3   use strict;
4   use warnings;
5
6   use Test::More;
7
8   use MyTestUtil ("fullpath", "chtempdir", "is_todolist_content", "make_datafile");
9
10  my $program_filename = "07_list_notyet_todo.pl";
11  my $program_fullpath = fullpath($program_filename);
12
13  {
14      # 準備
15      chtempdir();
16
17      make_datafile(join("",
18              "Done,some todos\n",
19              "some todos 2nd\n",
20          ));
```

```
21
22      # 実行
23      my $got = `$program_fullpath`;
24
25      # 結果比較
26      my $expected = join("",
27                          "2:some todos 2nd\n",
28                         );
29      is($got, $expected);
30  }
31
32  done_testing();
```

10.4.3 3

テストするプログラムを実行して入力する部分を関数にしました.

04_add_todo.t

```
1   #!/usr/bin/env perl
2
3   use strict;
4   use warnings;
5
6   use Test::More;
7
8   use MyTestUtil ("fullpath", "chtempdir", "is_todolist_content", "execute_with_input");
9
10  my $program_filename = "04_add_todo.pl";
11  my $program_fullpath = fullpath($program_filename);
12
13  {
14      chtempdir();
15
16      my $new_todo_content = "append new todo\n";
17      execute_with_input($program_fullpath, $new_todo_content);
18
19      my $expected = $new_todo_content;
20      is_todolist_content($expected);
```

```
}

done_testing();
```

06_done_todo.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use MyTestUtil ("fullpath", "chtempdir", "is_todolist_content", "make_datafile", "execute

my $program_filename = "06_done_todo.pl";
my $program_fullpath = fullpath($program_filename);

{
    # 準備
    chtempdir();

    make_datafile(join("",
            "some todos\n",
            "some todos 2nd\n",
        ));

    # 実行
    my $target = "1\n";
    execute_with_input($program_fullpath, $target);
    print("\n");

    # 結果比較
    my $expected = join("",
                        "Done,some todos\n",
                        "some todos 2nd\n",
                       );
    is_todolist_content($expected);
}

```

```
done_testing();
```

MyTestUtil.pm

```
package MyTestUtil;

use strict;
use warnings;

use Cwd;
use File::Temp ("tempdir");
use Test::More;

use Exporter ("import");
our @EXPORT_OK = (
    "fullpath",
    "chtempdir",
    "is_todolist_content",
    "make_datafile",
    "execute_with_input",
);

sub fullpath {
    my ($program_filename) = @_;
    my $pwd = cwd();
    return join("/", $pwd, $program_filename);
}

sub chtempdir {
    my $temporary_directory = tempdir(CLEANUP => 1);
    chdir($temporary_directory) or die; # 現在のディレクトリを一時ディレクトリにする
}

sub is_todolist_content {
    my ($expected, $message) = @_;

    open(my $fh, "<", "todolist.txt") or die;
    my $got = join("", <$fh>);
    close($fh) or die;

```

```
    is($got, $expected, $message);
}

sub make_datafile {
    my ($content) = @_;

    open(my $wfh, ">", "todolist.txt") or die;
    print($wfh $content);
    close($wfh) or die;
}

sub execute_with_input {
    my ($program_fullpath, $input) = @_;

    open(my $wfh, "|-", $program_fullpath) or die;
    print($wfh $input);
    close($wfh) or die;
}

1;
```

10.4.4 4

今回短いので，エクスポート (EXPORT) するのを省略しました.

03_init_todo.pl

```
#!/usr/bin/env perl

use strict;
use warnings;

use FindBin;
use lib ($FindBin::Bin);
use MyTodolist;

my $todo = "sample todo";
MyTodolist::init_todo($todo);
```

04_add_todo.pl

```
#!/usr/bin/env perl

use strict;
use warnings;

use FindBin;
use lib ($FindBin::Bin);

use MyTodolist;

MyTodolist::add_todo();
```

05_list_todo.pl

```
#!/usr/bin/env perl

use strict;
use warnings;

use FindBin;
use lib ($FindBin::Bin);

use MyTodolist;

MyTodolist::list_todo();
```

06_done_todo.pl

```
#!/usr/bin/env perl

use strict;
use warnings;

use FindBin;
use lib ($FindBin::Bin);

use MyTodolist;

MyTodolist::done_todo();
```

07_list_notyet_todo.pl

```perl
#!/usr/bin/env perl

use strict;
use warnings;

use FindBin;
use lib ($FindBin::Bin);

use MyTodolist;

MyTodolist::list_notyet_todo();
```

MyTodolist.pm

```perl
package MyTodolist;

use strict;
use warnings;

use Exporter ("import");
our @EXPORT_OK = ();

sub init_todo {
    my ($todo) = @_;
    open(my $fh, ">", "todolist.txt") or die;
    print($fh $todo, "\n");
    close($fh) or die;
}

sub add_todo {
    print("Input todo: ");
    my $content = <STDIN>;
    chomp($content);
    print($content, "\n");

    open(my $fh, ">>", "todolist.txt") or die;
    print($fh $content . "\n");
    close($fh) or die;
}
```

```
26
27  sub list_todo {
28      open(my $fh, "<", "todolist.txt") or die;
29      my $count = 1;
30      while (defined(my $line = <$fh>)) {
31          print($count, ":", $line);
32          $count = $count + 1;
33      }
34      close($fh) or die;
35  }
36
37  sub done_todo {
38      print("which number?: ");
39      my $num = <STDIN>;
40
41      open(my $rfh, "<", "todolist.txt") or die;
42      my @lines;
43      while (defined(my $line = <$rfh>)) {
44          push(@lines, $line);
45      }
46      close($rfh) or die;
47
48      open(my $wfh, ">", "todolist.txt") or die;
49      my $count = 1;
50      foreach my $line (@lines) {
51          if ($num == $count) {
52              $line = "Done," . $line;
53          }
54          print($wfh $line);
55          $count = $count + 1;
56      }
57      close($wfh) or die;
58  }
59
60  sub list_notyet_todo {
61      open(my $fh, "<", "todolist.txt") or die;
62      my $count = 1;
63      while (defined(my $line = <$fh>)) {
64          chomp($line);
```

```
65          my ($state, $content) = split(/,/, $line);
66          if ($state ne "Done") {
67              print($count, ":", $line, "\n");
68          }
69          $count = $count + 1;
70      }
71      close($fh) or die;
72  }
73
74  1;
```

それぞれの *.pl ファイルの中が関数呼び出しだけになっていてほとんど同じです．この関数呼び出しを分岐する関数を作ってしてしまえば，*.pl ファイルは 1 つだけにできるかもしれません．色々と考えてみるのもよいでしょう．

10.5 7 練習問題

10.5.1 1

@ARGV は，通常のリスト変数と同じように扱えばよいでしょう．

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5
6  my $name = "john";
7
8  if (@ARGV > 0) {
9      ($name) = @ARGV;
10 }
11
12 print("Hello, ", $name, "\n");
```

10.5.2 2

ex7_2.pl

```perl
#!/usr/bin/env perl

use strict;
use warnings;

if (@ARGV < 2) {
    exit;
}

my ($b, $e) = @ARGV;

my $sum = 0;
my $i = int($b);
while ($i <= $e) {
    $sum = $sum + $i;
    $i = $i + 1;
}
print($sum, "\n");
```

10.5.3 3

ex7_3.pl

```perl
#!/usr/bin/env perl

use strict;
use warnings;

use Ex7 ("sum");

if (@ARGV < 2) {
    exit;
}

my ($b, $e) = @ARGV;


print(sum($b, $e), "\n");
```

Ex7.pm

```perl
package Ex7;

use strict;
use warnings;

use Exporter ("import");
our @EXPORT_OK = ("sum");

sub sum {
    my ($b, $e) = @_;
    my $sum = 0;
    my $i = int($b);
    while ($i <= $e) {
        $sum = $sum + $i;
        $i = $i + 1;
    }
    return $sum;
}

1;
```

10.5.4 4

ex7_4.pl

```perl
#!/usr/bin/env perl

use strict;
use warnings;

use Ex7 ("sum");

if (@ARGV < 2) {
    exit;
}

my ($b, $e) = @ARGV;

```

```
14 if ($b =~ m/[^0-9.+-]/ or $e =~ m/[^0-9.+-]/) {
15     print("arg is not number: arg = (", $b, ", ", $e, ")\n");
16     exit;
17 }
18
19 print(sum($b, $e), "\n");
```

エラーになったときに，どういう入力をしていたか表示してあげてもよいですね．

ex7_4.t

```
1  #!/usr/bin/env perl
2
3  use strict;
4  use warnings;
5
6  use Test::More;
7
8  my $filename = "./ex7_4.pl";
9
10 {
11     my @arg_strs = (
12         "",
13         "a",
14         "a 1",
15         "1 10",
16         "-3.5 10",
17     );
18     my @expecteds = (
19         "",
20         "",
21         "arg is not number: arg = (a, 1)\n",
22         "55\n",
23         "49\n",
24     );
25     my $i = 0;
26     while ($i < @arg_strs) {
27         my $arg_str = $arg_strs[$i];
28         my $expected = $expecteds[$i];
```

```
        my $cmd = $filename . " " . $arg_str;
        my $got = `$cmd`;
        my $message = "args: " . $arg_str;
        is($got, $expected, $message);
        $i = $i + 1;
    }
}

done_testing();
```

複数のテストをするのにリスト変数を使って，入力，出力を管理するようにしました．

is_number() 関数版は，自分で書いてみましょう．

又，数字を表現するのには，もう少し厳密にチェックすることもできます．

m/[0-9.+-]+/ では，"+++" のようなものも OK になります．

例えば，以下がすぐに思い付きます．

- 先頭は，符号 (+ や -) が来ることがある．
- 先頭の数字は，0 ではない (= 1 ‖ 9)
  - 小数点以下の場合や 0 の場合には，その限りではない
- .(ドット) は，1 回までしか出現しない
- 空文字ではない
- 小数点の整数部が 0 の場合は省略できる

問題となるパターンを上げると

- "100-"
- "010"
- "1..1"

です．これらを考えると，m/\A[-+]?(([1-9][0-9]*|0)(\.[0-9]*)?|\.[0-9]+)\z/ この辺りが妥当でしょうか？

意味毎のパーツに分けると以下のようになります．色々入力してみて試してみるのも良いでしょう．

```
m/
      \A
        [-+]?
        (
```

```
          (
            [1-9][0-9]*
            |
            0
          )
          (\.[0-9]*)?
          |
          \.[0-9]+
        )
      \z
/x
```

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

{
    my $re = qr'
      \A
        [-+]?
        (
          (
            [1-9][0-9]*
            |
            0
          )
          (\.[0-9]*)?
          |
          \.[0-9]+
        )
      \z
    'x;
    my @ng_strs = (
        "",
        "100-",
        "010",
```

```
        "1..1",
    );
    foreach my $str (@ng_strs) {
        ok(not($str =~ m/$re/), "F: str = " . $str);
    }
    my @ok_strs = (
        "0",
        "0.",
        "0.1",
        ".1",
        "10.1",
    );
    foreach my $str (@ok_strs) {
        ok($str =~ m/$re/, "T: str = " . $str);
    }
}

done_testing();
```

こんな長い正規表現はあまり書きません．練習として解析してみるのもよいでしょう．

10.5.5 5

以前に載せてあるので省略します．

10.5.6 6

ex7_6.t

```
#!/usr/bin/env perl

use strict;
use warnings;

use Test::More;

use Ex7;
```

```
9
10  {
11      my @bs = (
12          1,
13          -3.5,
14          10,
15      );
16      my @es = (
17          10,
18          10,
19          1,
20      );
21      my @expecteds = (
22          55,
23          49,
24          0,
25      );
26      my $i = 0;
27      while ($i < @bs) {
28          my $b = $bs[$i];
29          my $e = $es[$i];
30          my $got = Ex7::sum($b, $e);
31          my $expected = $expecteds[$i];
32          my $message = join("",
33              "Ex7::sum(", $b, ", ", $e, ") = ", $expected,
34          );
35          is($got, $expected, $message);
36          $i = $i + 1;
37      }
38  }
39
40  done_testing();
```

11　付録

11.1　補足: モジュールのドキュメントの探し方

- http://perldoc.jp/

ここに和訳済みのものがいっぱいあります.

ここに無ければ，英語のものを探す必要があるかもしれません.

- http://search.cpan.org/

ここを探すとよいでしょう.

又，perldoc コマンドで調べることもできます．英語なのですが，簡単に調べることができます.

```
$ perldoc Test::More      # モジュールを調べたいとき
$ perldoc -f open         # 関数を調べたいとき
$ perldoc perlop          # まとまった項目 (perlop = 演算子)
```

12　さいごに

これを読んだ人がプログラミングを挫折しないで学べるようになることを望んで書きました.

なるべく覚えることを減らして，1 つの章でポイントを絞って書くようにしました．その代わり網羅性は考えないことにしました．一般のプログラミング言語の入門書に通常あるようなことでも書いていないことがあります．10 個のことを覚えるより，1 つのことを 10 回やった方がうまくなると思っているからです.

勉強した方の生活がどこか楽になってくれると嬉しく思います.
