Perl によるプログラミング入門 in 東海

第一回

2016/02/20 第一版

目 次

---

1 はじめまして

1.1 はじめに

ここに来たということは，プログラミングを勉強しようと思って来たと思っています．プログラムで何かやろうと思っていることがあるかもしれませんし，まだそこまでイメージできていないけど，とりあえず勉強しよう．ということかもしれません．

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 Perl について

2.1 Perl 言語

幸か不幸か，ここではプログラミング言語に Perl を選択しています．少しだけ Perl 言語についてお話しします．

Perl 言語は，3 分間で書き上げるようなやっつけ仕事にも，本格的なプログラムにも適しています．特に，テキスト処理が 90%，その他が 10% くらいの処理を書くのに適しています． 逆に，バイナリデータ [1] の処理は不得意です．

Perl は無料で手に入ります．

[1] バイナリデータとは，テキストではないデータです．つまり人が読めない形式のデータです．

3　最初のプログラム

まずは，プログラムを書いてみて雰囲気を掴みましょう．

3.1　プログラムを書いて，実行してみる

- 作業用ディレクトリ (フォルダ) を作る
  - ディレクトリ名: `perl20160220`
- テキストエディタを開く
  - テキストエディタで作業フォルダ内にファイルを作る
  - ファイル名: `01_hello.pl`
- コンソール (コマンドプロンプト) を出して作業フォルダに移動する
  - 作業フォルダ内のファイルを実行する
  - 例: `perl 01_hello.pl`
- 結果，コンソールに Hello, World と表示されたことを確認する

3.1.1　実行例

```
$ mkdir perl20160220
$ cd perl20160220
$ edit 01_hello.pl
$ perl 01_hello.pl
```

3.1.2　01_hello.pl

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

use strict;
use warnings;

print("Hello, World\n");
```

3.2 解説 - 周辺

まず，プログラムを実行するには，どのプログラムファイルを実行するかを指定しなければなりません．どこにプログラムファイルを置いたかすぐわかるように，今日一日は，同じ作業ディレクトリで作業するようにしましょう．

`perl 01_hello.pl` という呪文でプログラムを実行しました．これは，`perl` というコマンド (命令) を使って，`01_hello.pl` という Perl 言語で書かれたプログラムファイルを実行した．ということです．`perl + プログラムのファイル名`という形式は今後も使いますので，慣れておいてください．

では，プログラムファイルの解説に移りましょう．

3.3 解説 - プログラムファイル

Perl 言語は，上から順番に処理が進む，逐次処理を基本とします．プログラムファイルは，テキストファイルです．ソースコードやソース，ソースファイルとも呼ばれます．Perl 言語で書かれたソースコードの中では，`#` (シャープ) から行末まではコメント (説明文) の意味になります．コメントは，プログラムとしては何も実行されず，意味がありません．ですが，プログラムが何をしているかをソースコードを読む人に説明するために使います．

```
1 #!/usr/bin/env perl     # <= Perl で動作するようにシェルに伝える
2
3 use strict;   # 文法を厳密にチェックする
4 use warnings; # 警告を出す
5
6 print("Hello, World\n"); # 端末に文字列を出力する
```

最初の行は，Mac, Linux の人には意味があります．先程，`perl + プログラムのファイル名` という形式で，Perl 言語のプログラムファイルを実行すると言いましたが， 実は，Mac や Linux の人には必要ありません． このプログラムファイルは，`perl` で実行する．というのはこの行に書かれています．

Windows の人は，ファイル名の拡張子部分で判定するようになっているので，この部分は意味がありません．

それ以外の環境の人は，その環境に詳しい人に聞いてください．

次の行の，`use strict; use warnings;` の部分は，警告を多めに出すように指定してあります．慣れないうち，慣れていても，情報が多い方が何かがあったときに気付きやすいので，この 2 行は基本的には書きましょう．

ここまでは，プログラムの動作にはあまり関係ありませんでした．残りの 1 行が，このプログラムファイルの本体です．

この行は，`print()` という関数を使って，コンソールに出力しています．プリントのくせに，プリンタへの出力ではありません．出力した内容は，`"Hello, World"` であって，`"Hello, World\n"` ではありません．`\n` はどこへ行ってしまったのか．その部分を削除するなどして確かめてみてください．

意外に重要なのが，`;` (セミコロン) で，1 つの文の区切りを意味します．

特に説明しませんでしたが，空行 (からぎょう) といって，行に空白文字やタブ文字以外何も書いていないものも，プログラムとして何もしません．コメントと同様に，プログラムファイルを読む人に意味上の区切りを表現するために使うことが多いようです．この会の中に出てくる例でも，そのように使います．日本語の文章で言うところの，段落，のようなものでしょうか．

3.4 練習問題

1. 一人でやってみよう
2. 他にも何かを表示してみよう (名前，年齢)
3. `perldoc -u -f atan2` を実行してみよう
4. 以下のソースコードをファイルに保存して，実行して，結果を比較してみよう．何が変わりましたか？

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

use strict;
use warnings;

my @lines = `perldoc -u -f atan2`;
foreach (@lines) {
    s/\w<([^>]+)>/\U$1/g;
    print;
}
```

4 まずはプログラミングに慣れる

プログラミングといってもあまり怖がる必要はありません．簡単なところから初めましょう．

4.1 簡単な計算

プログラミング言語では，数値計算ができることが多いです． Perl でも，四則演算など基本的な計算は可能です．

4.1.1 02__calc.pl

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

use strict;
use warnings;

print(1 + 1, "\n");  # => 2

my $x = 10;
print($x * $x, "\n"); # => 10 * 10 => 100
```

4.2 解説 - 値 と変数

最初の行は，`1 + 1` を計算した結果を，コンソールに出力しています．`print()` は，コンソール出力をする命令でした． `1 + 1` は，ご存知の通り，数値計算です． `+` という演算子は数値と数値を足して，新しい数値を作ります．この場合は，新しい数値は，`print()` 文によってコンソールに出力しています．

次の文 `my $x = 10` は，`$x` という変数に数値 という値を代入しています．代入というのは値 に，名前を付けることです． この場合，名前は`$x` です． 代入の前に `my` が付いています．これは今はあまり気にしないでください．変数を最初に使う場所で使います．

`10` に `$x` という名前を付けました． 次に `print($x * $x, "\n")` という文で，`$x` 掛ける `$x` を計算しています．`$x` は `10` でしたので，`10 * 10` の計算と同じ意味になります．

4.3 単語 - 値，変数

4.3.1 値，スカラ

新しい言葉がいくつか出てきたので，まとめておきます．

値 とは，Perl で扱える最小単位のデータです． スカラとも言います．

スカラ には，文字列と数値 があります．スカラの種類はそれ以外にもありますが，一番使うこれらを覚えておけばよいでしょう．

文字列 を作るには，以前出てきたように，ダブルクォートを使います．数値を作るには，0 から 9 と . (ドット) などから作ることができます．

- スカラ．値
  - 文字列
    - 例. `"\n"`, `"Hello, World\n"`
  - 数値
    - 例. `1`, `2.5`

4.3.2 演算子

Perl では，数学の演算子をいくつか使うことができます． `+`, `-`, `*`, `/` です．掛け算，割り算の記号はなじみがないかもしれませんが，こういう記号を使います．

四則演算は，全て 数値 に対しての演算でした． 例えば，`+` (足し算) は，2 つの数値 を使って，1 つの数値を得るための演算です．

文字列 に作用する演算子もあります． `.` (ドット) です．これは 文字列 と 文字列 を結合して，新しい文字列 を作ります．

```
1 print("abc" . "def", "\n"); # => abcdef が表示される．
```

文字列 と 数値 の両方使う演算子もあります． `x` は，文字列の繰り返しの意味の演算です． 文字列と繰り返しの数の数値 を使って，新しい文字列 を返します．

```
1 print("a" x 3, "\n"); # => aaa が表示される
```

- 演算子．
  - 数値，数値 → 数値
    - 四則演算: `+`, `-`, `*`, `/`

  - 文字列，文字列 → 文字列
    * 文字列連結: . (ドット)
  - 文字列，数値 → 文字列
    * 文字列の繰り返し: x

4.3.3 関数呼び出し

print() について少し解説します． print() は，コンソールに出力する命令でした． このような命令は，関数と呼ばれます．関数は，関数名 + "(" + 引数 1 + "," + 引数 2… + ")" の形式で呼び出されます．今後もいくつか関数が出てきますが，基本は同じ呼び出し方です．(例外もあります)

print() は，すべての引数を連結してコンソールに出力します．

- 関数呼び出し
  - 例: print(1 + 1, "\n")

4.3.4 スカラ変数

変数は，値に名前を付けるための機能です．最初に変数を使うときには，my を使います．特にスカラに関連させる変数をスカラ変数と呼びます．スカラ変数は，"$" + 変数名 の形式にします．

my には，変数の有効期間を設定する機能もありますが，今はあまり意識しなくともよいでしょう．

- 変数
  - my で宣言する
  - スカラ変数
    * $ + 変数名 の形式で使う
    * 変数 = 値 で，変数に値をひもづけます．
      · 例: $num = 10, $greeting = "おはよう"
    * 変数 の形式で変数にひもづいた値を使えます
      · 例: $x = 10; $x + 1; # => 11

4.4 練習問題

1. いくつか計算してみましょう．例 1 + 2 * 3．答えは思い通りになりましたか？

2. 半径 12.5 の円の円周の長さを求めるプログラムを書いてください．円周の長さは半径の 2 π 倍 です．答えは，78.5 くらいになる筈です．(半径は，`$r` というスカラ変数に入れてみてください． π は 3.14 として計算すると良いでしょう．)

5　今回目指すもの

この会では，TODO リストを作りながら，一緒にプログラミングについて学びます．まず，何を作るかです．

5.1　今回目指すもの

TODO リストプログラムを作ります．どんな機能があるでしょうか？ 私達の作る TODO リストは，以下のような機能を持っています．

- TODO リストとは，TODO を保存，操作します
- TODO の状態は，未実施，完了 があります．
- TODO には内容 (content) があります．

5.2　練習問題

- 自分でも TODO リストに欲しい機能を考えてみましょう

6　TODO リストを保存する

ではまず，固定の TODO リストを作成するプログラムを作ってみましょう．

6.1　03__init__todo.pl

```
1 #!/usr/bin/env perl
2
3 use strict;
4 use warnings;
```

```
5
6  open(my $fh, ">", "todolist.txt") or die;
7  print($fh "sample todo\n");
8  close($fh) or die;
```

新しい関数が出てきました．順に説明します．

6.2 open() / close() 関数

ファイルの中身を読み込んだり，書き込んだりするための準備や後片付けが必要です．それを行うのが，open()，close() 関数です．

open() 関数は，引数 が 3 つあります．

- 1 つめは，ファイルハンドル，my $fh の部分です．ファイルハンドルというのは，ファイルを扱うためのスカラ値です．print() 関数のところでもう少し説明します．
- 2 つめは，モードです．書き込みか読み込みかを文字列で指定します．
  - ">" は書き込み
  - "<" は読み込みで，新規ファイルを作って先頭から書き込みます．
  - ">>" は，追記書き込みで，ファイルの末尾に追記します．
- 3 つめは，ファイル名です．
  - ファイル名だけ書くと，プログラムを実行したディレクトリと同じ場所にあるものになります．

close() 関数は，ファイルハンドルの後片付けをします．忘れないようにしましょう．

ファイルの書き込み読み込みは，以下のような形が基本です．覚えてしまいましょう．

書き込み

```
1  open(my $fh, ">", "filename") or die;
2  # 何か処理をする
3  close($fh) or die;
```

読み込み

```
1  open(my $fh, "<", "filename") or die;
2  # 何か処理をする
3  close($fh) or die;
```

`or die` については，ちょっと難しいので，とりあえずこういうものとして覚えておいてください．

6.3 print() 関数

`print()` です．これはコンソールに出力する関数だった筈です．実は，それ以外の機能もあります．ファイルハンドルを指定すると，そのファイルに出力してくれます．このときに，ファイルハンドルを指定した後には，カンマがないことに注意してください．

```
print("Hello, World\n");

print($fh "Hello, World\n");  # カンマが無いことに注意
```

6.4 練習問題

1. todolist.txt ファイルの内容を確認してみましょう．
2. ファイルに色々書き込んでみましょう．

7 TODO を追加する

さっきの例では，1 つのプログラムから固定の TODO しか作れませんでした．こんどは，プログラムを変えないで，新しい TODO を追加する機能を作りましょう．

7.1 04_add_todo.pl

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

use strict;
use warnings;

my $content = <STDIN>;
chomp($content);
print($content, "\n");
```

```
9
10 open(my $fh, ">>", "todolist.txt") or die;
11 print($fh $content, "\n");
12 close($fh) or die;
```

7.2 行入力演算子

<> (この場合だと<STDIN>) は，演算子です．今までに出てきたものとちょっと使い方が違います．

今までの演算子の使い方の例．

```
1 1 + 2;      # => 3 : 数値, 数値 → 数値
2 "a" x 3;    # => aaa  文字列, 数値 → 文字列
```

今までに出てきた演算子は，全て演算子の両側に引数を取っていました．<> は，< + ファイルハンドル + > のように真ん中に引数を書きます．

ですので，<STDIN>の引数は，STDIN です． STDIN はファイルハンドルです．open() 関数のところで出てきたものと同じです． STDIN は，コンソールからの入力が溜められているファイルハンドルです．

```
1 my $line;
2
3 $line = <STDIN>;
4 $line = <$fh>;
```

- ファイルハンドル → 文字列
  - 行入力演算子：<>

7.3 スカラの自動変換

Perl は，スカラ値の自動変換が行われます．

```
1 # 文字列 → 数値 への自動変換 ("10" → 10)
2 1 + "10"; # => 11
3
4 # 数値 文字列の自動変換 (3 → "3")
5 3 x 4; # => 3333
```

Perl がうまくやってくれるので，こういうことが起きている．ということだけ知っておいてください．

7.4 chomp() 関数

`chomp()` は，末尾の改行を取り除きます．変数を指定すると，その変数にひもづいている値が変更されます．

7.5 練習問題

1. プログラムを実行してみてファイルの中身を見て TODO が追記されていることを確認しましょう
2. 何個かの入力をして，計算をするプログラムを書いてみてください．例えば，2 つの数を入力してもらって，足し算をするプログラムなどです
3. 半径を入力したら，直径を表示するプログラムを作ってください
4. 下のようなファイルをそれぞれ入力して，出力するプログラムを作ってみてください

入力 (ex7_4_in.txt)

```
3
4
```

出力 (ex7_4_out.txt)

```
7
12
3333
```

8 現在 の TODO を表示する

TODO を追加するところまでできるようになりました．では，現在の TODO 全部を表示できるようなプログラムを作りましょう．

8.1 05_list_todo.pl

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

use strict;
use warnings;

open(my $fh, "<", "todolist.txt") or die;
my $count = 1;
while (defined(my $line = <$fh>)) {
    print($count, ":", $line);
    $count = $count + 1;
}
close($fh) or die;
```

8.2 defind() とファイルの終端

<> 行入力演算子は，ファイルの終端まで入力が進むと，undef という値を返す． これもスカラ値です．

defined() 関数は，引数が undef かどうか判定してくれて，真偽値を返してくれます． 真偽値については，次の while の条件部分のところで触れます．

8.3 while 文

while というのは，とある条件が満たされている間に，何か別のことを行う．というときに使います．

例えば，while (A) { B } のようになっていたら，最初に A が満たされているかチェックし， 満たされていれば，B を処理し，次に A をチェック，B を処理… で A が満たされなくなったときに，この繰り返しが終了し，次の行に処理が移動します．

条件が満たされている状態を真，そうでない状態を偽と呼びます． Perl では，偽とはスカラ値で言うと，undef，数値の 0，文字列の "" (空文字列) です． 又，文字列"0" は，数値 0 に自動変換されるので，これも偽です． "" 空文字列に自動変換される数値はありませんので，逆はありません． 真は，それ以外で表現できます．

- ブーリアン (スカラ版)

  - 偽：undef, 0, "", "0"
  - 真：それ以外

例えば，

```
while (1) {
    print("hello", "\n");    # 実行する前に説明を読んでね
}
```

この場合は，条件部分が 1 となっていて，これは数値の 0 ではないので， 常に真 です．つまり print() 文を処理し，条件に戻っても，常に真なので，print() が無限に繰り返されることになります．

無限に繰り返すのはあまり良くないので，繰り返しの回数を決めて処理することがよくあります．

例えば，

```
my $count = 1; # 最初の値
while ($count < 10) { # 繰り返しは，count が 10 になったら終了する
    print($count, "\n");
    $count = $count + 1; # 繰り返しのたびに値を更新している
}
```

こうすると，最初は $count が 1 ですが，ブロックの中の$count = $count + 1 が実行されるたびに $count の値が 1 つずつ増えていくので，$count が 10 になったら，繰り返しが終了します．この処理はよくあるので覚えておくと良いでしょう．

8.4 ブーリアン演算子

ブーリアンを扱う演算子がいくつかあります．例えば数値に対してだと，<, > などの記号は見たことがあるのではないでしょうか？これらは，ご存知の通りに数値の大小でブーリアンとして適切なスカラ値を返します．

- 演算子
  - 数値，数値 → ブーリアン [2]
    - <, >, <=, >=, ==, !=
  - 文字列，文字列 → ブーリアン
    - eq, ne

[2] 正確には，ブーリアン値というスカラ値は存在しないのですが便宜上ブーリアン値があるように記述とさせて頂きます．

- ブーリアン，ブーリアン → ブーリアン
  - and, or, &&, ||
- ブーリアン → ブーリアン
  - not, !

8.5 練習問題

1. ファイルの中身を読み込んで，全部コンソールに出力するプログラムを書いてみましょう
2. 1 から 10 までの数を表示するプログラムを書いてみてください．(ヒント．`$n = $n + 1` をブロックの中で使うと，数を順に進めることができます．又，10 まで，というのは，10 以下である間，と考えてもいいかもしれません．)
3. 1 から 10 までの数を足した結果を表示するプログラムを書いてみてください．(ヒント．足した結果を保持するスカラ変数，`$sum` を使うといいかもしれません．)
4. 数字 (n) を入力して，1 から n までを足した結果を出力するプログラムを書いてみてください．
5. 20 未満の数で，1, 4, 7, … のように，3 ずつ離れた数を出力するプログラムを書いてみてください．
6. “quit” という文字列を入力しないと終わらないプログラムを書いてみてください．

9 TODO を完了済みにする

TODO を完了させられなければ，TODO リストとしては未完成です．TODO を指定して，その TODO を完了済み (Done) としてマークできるようにしましょう．

9.1 06_done_todo.pl

```
1 #!/usr/bin/env perl
2
3 use strict;
4 use warnings;
```

```
5
6  print("which number?: ");
7  my $num = <STDIN>;
8  chomp($num);
9
10 open(my $rfh, "<", "todolist.txt") or die;
11 my @lines;
12 while (my $line = <$rfh>) {
13     push(@lines, $line);
14 }
15 close($rfh) or die;
16
17 open(my $wfh, ">", "todolist.txt") or die;
18 my $count = 1;
19 foreach my $line (@lines) {
20     if ($num == $count) {
21         $line = "Done," . $line;
22     }
23     print($wfh $line);
24     $count = $count + 1;
25 }
26 close($wfh) or die;
```

9.2 リスト / リスト変数

単一のデータとしてスカラ値があるという話をしました．複数のスカラ値をまとめて取り扱うために，リストという値があります．

```
1 (1, 2, 3);
```

リストは，() (括弧) で囲まれて，, (カンマ) で区切られたスカラ値で表 現します．

スカラにスカラ変数があったように，リストにもリスト変数があります．リスト変数は，@ + 変数名 の形式です．スカラ変数と同じように，my で宣言して，= で代入できます．

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

リストには，複数のスカラ値が含まれています．個別のスカラ値を取り扱うために，[] を使って，添字で位置を指定します．位置は 先頭が 0 番です．

```
my @numbers = (1, 2, 3, 4, 5);
$number[0];    # => 1
$number[2];    # => 3
```

リスト変数全体を表すときには，先頭の記号が，@ ですが，リストの中のスカラ値部分を扱う場合には， 先頭の記号が$ になります．スカラとして扱うときは$，リストとして扱うときは @ として覚えてしまいましょう． [] 内の数値 (添字) は，リストの先頭からの位置で場所を指定します．先頭の位置は，0 です． リストの一部を代入して変更することもできます．

```
my @numbers = (1, 2, 3, 4, 5);
$number[0] = 10; # => @numbers は (10, 2, 3, 4, 5)
```

- 変数
  - リスト変数
    * リストとして扱うとき: @ + 変数名
    * スカラとして扱うとき: $ + 変数名 + [+ 位置 +]

9.3 push() 関数

リストを操作する関数の一つです． リスト変数の最後にに要素を追加します．

```
my @numbers; # @numbers => ()

push(@numbers, 1); # @numbers => (1)
push(@numbers, 2); # @numbers => (1, 2)
push(@numbers, 3); # @numbers => (1, 2, 3)
```

9.4 if 文

while と同じように，条件とブロックがあります． while と違い繰り返したりしません．条件式が真であればブロックを実行，そうでなければ，ブロックを実行しません．

```
my $input = <STDIN>;
if ($input) {
    print("Your input(", $input, ") is true", "\n");
}
```

9.5 練習問題

1. alpha bravo charlie delta echo のような文字列のリストを作り，それを順に出力するプログラムを書いてください．
2. 半径を入力して直径を出力するプログラムを作りましたが，半径が負の数だったら，直径として 0 を表示するようにしてください
3. 以下のプログラムがどうなるか予想して，実際に確認してみてください

```
if ("0") {
    print("I heard truth", "\n");
}
```

4. 例えば以下のようなファイルを自分で作成して，そこから数字取得して，全部を合計してコンソールに出力するプログラムを書いてください．入力ファイルは各行に 1 つずつ数字があるようにしてください．

ex9_4_in.txt

```
1
3
8
10
```

10 TODO の内容のうち未完了のものだけ表示する

終了した TODO は見る必要が無いと思います． では，TODO のうち未完了のものだけ表示するプログラムを作りましょう．

10.1 07_list_notyet_todo.pl

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

use strict;
use warnings;

open(my $fh, "<", "todolist.txt") or die;
my $count = 1;
while (my $line = <$fh>) {
    chomp($line);
```

```
    my ($state, $content) = split(/,/, $line);
    if ($state ne "Done") {
        print($count, ":", $line, "\n");
    }
    $count = $count + 1;
}
close($fh) or die;
```

10.2 split() 関数

split() は，区切り文字と文字列を引数に取り，文字列を分割します．

文字列を分割して，文字列のリストを返します．

10.3 スカラコンテキスト / リストコンテキスト

コンテキストは，その位置に置かれたものは，何として扱われるか，ということです．言葉だけではイメージできないと思いますので，例を上げます．

```
# スカラーコンテキスト
my $line = <>;

# リストコンテキスト
my ($line1, $line2, $line3) = <>;
```

<> 行入力演算子に何を求めているか，でコンテキストが違う．と表現します．例えば，上の例で my $line = <> の方は，代入の相手がスカラ変数なので，<> もスカラを返そうとします．これがスカラコンテキストです．又，my ($line1, $line2, $line3) = <> の方は，代入の相手はスカラではありません．<> に複数の値を求めています．リストコンテキストと呼ばれます．

見分け方としては，代入先が () (括弧) を使っているか，リスト変数か否かです．

```
my ($line) = <>;
```

これは，最初の例と同じように見えますが，() を使っているのでリストコンテキストになります．

では，コンテキストが違うということがわかったところで，<> は，コンテキストによってどういう違いがあるかというと，スカラコンテキストでは，1 行だけ結果を返していたのが，リストコンテキストでは，全部の行をリストで返します．

では，以下のような場合，入力が 3 行以上あったとすると，どうなるでしょうか？

```
1 my ($line1, $line2) = <>;
```

答は，3 行目以降は捨てられます．逆に入力が足りなかった場合には，undef で埋められます．例えば 1 行しか入力が無かったら，$line2 は undef になります．

上記の例は，代入でしたが，while などの条件式や演算子の引数もスカラコンテキストです．

- コンテキスト
  - スカラコンテキスト
    - スカラ変数への代入，条件式，演算子の引数部分
  - リストコンテキスト
    - それ以外

10.4 練習問題

1. 例えば，“2014,10,20” のような文字列を入力したら，“2014 年 10 月 20 日です” という文字列を出力するプログラムを書いてみてください．
2. 1 行に 1 つずつの数のリストを入力の終りまで読み込んで，以下に示す人名のリストから数に対応する人名を表示するプログラムを書いてください．例えば，入力された番号が 1, 2, 4, 2 だとすると，fread, betty, dino, betty となります．

   fred betty barney dino wilma pebbles bamm-bamm

11 まとめ

TODO リストを操作するプログラムを作ってみました．そのプログラムの解説をすることで，Perl プログラムの基礎を学びました．

プログラミング言語の基本的な性質について学びました．

- データ，データ構造
- 演算子
- 分岐 (if)
- 繰り返し (while)
- ファイルの書き込み，読み込み

TODO リストの機能の最低限のものは作りましたが，まだ機能としては不十分です．また引き続きお付き合いして頂くか，自分で改良してみてください．

12 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．

12.1 準備

- 作業ディレクトリを作って，そこで作業した
  - ファイル名を指定すると，作業ディレクトリのファイルを指定したことになる
- テキストエディタで Perl のプログラムファイルを作ることができる
- Perl プログラムファイルをコンソールから実行できる

12.2 基礎文法

- ファイル名は，.pl で終わるようにする
- # (シャープ) から行末までは，コメントとして扱われてプログラムとし ては実行されない
- 行の区切りに ; (セミコロン) を付ける
- 下記の基本の行を必ず書く

```
1 #!/usr/bin/env perl
2
3 use strict;
4 use warnings;
```

12.3 スカラ値

- 1 つ分のデータのことをスカラと呼ぶ
- スカラにはいくつかの種類がある
  - 数値
  - 文字列
  - ファイルハンドル
  - undef

```
# 数値の例
1
10
2.5
-100

# 文字列の例
"文字列"
"\n"
"Hello, World"

# ファイルハンドルの例
open(my $fh, "<", "filename") or die; # この内の $fh に入っている
STDIN

# undef の例
my $var;  # 変数を宣言だけして，代入する前には undef になっている
$var = undef;
```

又，場所によってスカラ値はブーリアン，真偽値としても扱われる．

- ブーリアン，真偽値
  - 偽: 0, "", undef, "0"
  - 真: それ以外

```
# 偽
if (0) {
    print("I said truth", "\n");
}
if ("") {
    print("I said truth", "\n");
}
```

```
8  if (undef) {
9      print("I said truth", "\n");
10 }
11
12 # 真
13 if (1) {
14     print("I heard truth", "\n");
15 }
16 if ("a") {
17     print("I heard truth", "\n");
18 }
```

12.4 リスト値

複数のスカラをまとめて扱う.

12.5 変数

12.5.1 スカラ変数

- スカラ変数はスカラに名前を付ける
- 最初に使うときには `my` をつける
- スカラ変数にスカラをひもづけるときには，`=` (イコール) を使う.
  - 代入と言う

12.5.2 リスト変数

- リスト値に名前を付ける
- 最初に使うときには，`my` をつける
- 代入には，`=` を使う
- それぞれのスカラ値を取り扱うときには，`[]` を使う
  - 例: `$ary[0]`, `$ary[0] = 10`

12.6 演算子

引数を取って，新しい値を返す．

- 数値，数値 → 数値
  - 四則演算: +, -, *, /
- 文字列，文字列 → 数値
  - 文字列連結: .
- 文字列，数値 → 数値
  - 文字列繰り返し: x
- ファイルハンドル → 文字列
  - 行入力演算子: <>
- 数値，数値 → ブーリアン
  - <, >, <=, >=, ==, !=
- 文字列，文字列 → ブーリアン
  - eq, ne
- ブーリアン，ブーリアン → ブーリアン
  - and, or, &&, ||
- ブーリアン → ブーリアン
  - not, !

12.7 関数

関数は，関数名 + "(" + 引数 1 + ", " + 引数 2 ... ")" の形式で呼び出す．

print()

引数をコンソールに出力する．最初の引数をファイルハンドルにすると，そのファイルハンドルに出力する

open()

指定したファイルのファイルハンドルを作る． "<" 読み込み，">" 書き込み，">>" 追記のモードがある．

close()

ファイルハンドルの後片付けをする．

chomp()

引数の値の末尾の改行文字を削除する．

push()

リスト変数の末尾に値を追加する．

`split()`

文字列を区切りで区切って，リスト値にして返す．

12.8 if 文

`if` は，ブロックを実行するかしないかを，条件式で決めます．

```
my $a = 10;
if ($a > 5) {
    print("over", "\n");
}
```

12.9 while 文

`while` では繰り返し処理を行うことができます．条件式の中は，ブーリアンとして解釈され，条件式が真の間，ブロックの中を実行し続けます．

```
my $count = 0;
while ($count < 5) {
    print($count, "\n");
    $count = $count + 1;
}
```

12.10 コンテキスト

コンテキストによって，動作を変える演算子，関数がある．

- コンテキスト
  - スカラコンテキスト
    - スカラへの代入，条件式部分，演算子の引数
  - リストコンテキスト
    - それ以外

例えば，以下があります．

- <> 行入力演算子
  - スカラ: 1 行分だけ返す

– リスト: 全部の行をリストにして返す

13 練習問題の答え

13.1 3. 練習問題

13.1.1 3.2

print() を次の行に書くことができます.

ex3_2.pl

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

use strict;
use warnings;

print("tetsu", "\n");
print("17", "\n");
```

このように 1 行に書いても同じように表示することができます.

```
print("tetsu\n17\n");
```

13.1.2 3.3

perldoc というコマンドは，Perl 言語のドキュメント (解説，説明) をコンソールに表示させることができるコマンドです． (残念ながら英語です)

atan2() という関数の説明を表示しています.

13.1.3 3.4

説明していないものがいくつか出ていますが，説明をします.

\‘(バッククォート) この演算子は，囲まれた部分をコマンドとして実行して，その出力結果を返します． この場合は，perldoc -u -f atan2‘ を実行します. 実行結果は，見た通りです.

\‘の結果を，@lines‘ 変数に保持しています．

foreach 文は，while のように繰り返しをします．真偽値による条件ではなく，繰り返す回数は，指定されたリスト値の回数です．foreach で，変数を指定しなければ，$_ がリスト値の 1 つずつに入った状態で，ブロックの中の繰り返しの処理を行います．

s/// は，文字列の置き換えです．説明していませんが，正規表現 を使って，指定された文字列の中の一部を置き換えます．この場合は，X<aaa> な文字列を AAA のように置き換えています．

print は，print() 関数のことで，何も指定しないとデフォルトで，$_ 変数の内容を出力することになっています．

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

# いつもの
use strict;
use warnings;

# atan2() という関数の説明を @lines というリスト変数に保持している
my @lines = `perldoc -u -f atan2`;

# @lines 変数の中身の各行を $_ というデフォルト変数に保持し，以下を繰り返す
foreach (@lines) {
    # $_ 変数の中の，何か 1 文字 + "<" + (">" 以外の文字 1 つ以上の連続) + ">" の部分を，さっき
    s/\w<([^>]+)>/\U$1/g;
    # $_ 変数の中身をコンソールに出力する
    print;
}
```

13.2 4. 練習問題

13.2.1 1.

出てきた演算子は，+, -, *, /, ., x です．それぞれを使ってみて，演算子に慣れましょう．

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

use strict;
```

```
use warnings;

print(1 + 3, "\n"); # => 4
print(3 - 1, "\n"); # => 2
print(2 * 5, "\n"); # => 10
print(5 / 2, "\n"); # => 2.5
print("aaa" . "bbb", "\n"); # => "aaabbb"
print("Abc" x 3, "\n"); # => "AbcAbcAbc"

print(1 + 2 * 3, "\n"); # => 7
```

13.2.2 2.

変数を使うときに最初に，my を入れるのを忘れないでください．

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

use strict;
use warnings;

my $r = 12.5;
my $pi = 3.14;
print(2 * $pi * $r, "\n"); # => 78.5
```

13.3 5. 練習問題

13.3.1 1.

TODO リストとして良くあるのは，期限やカテゴリです．良い機能があれば教えてください．

13.4 6. 練習問題

13.4.1 1.

print() で書き込んだ内容がファイルの中にあることを確認してください．

13.4.2 2.

どんな内容でも良いですが，以下のようになると思います． `open()` の後に `print()` があって，`close()` の順序になっていれば良いでしょう．

あと，書き込んだファイルの中身が想像通りになっているかどうかを確認してください．

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

use strict;
use warnings;

open(my $fh, ">", "sample.txt") or die;
print($fh "This is test file.", "\n");
print($fh "書き込まれているでしょうか？", "\n");
close($fh) or die;
```

13.5 7. 練習問題

13.5.1 1.

実行結果を確認してみてください． 改行が入ってしまっていませんか？ちゃんと入力した TODO が意図通りに追加されていましたか？

13.5.2 2.

これは，入力した値を使って演算を行う例です．

以下の例だと，演算子の期待している入力になっていないと警告が出るかもしれません．例えば，`+` は，数値を期待しているので，“a” などを入力すると，`Argument "a" isn't numeric in ...` のような警告が出てしまうでしょう．これらを回避する方法もありますが，一旦は無視します．

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

use strict;
use warnings;

```

```perl
my $op1 = <STDIN>; # 入力を受け取る
chomp($op1); # 入力された改行文字を削除する

my $op2 = <STDIN>;
chomp($op2);

print($op1 + $op2, "\n");
print($op1 - $op2, "\n");
print($op1 * $op2, "\n");
print($op1 / $op2, "\n");
print($op1 . $op2, "\n");
print($op1 x $op2, "\n");
```

13.5.3 3.

半径を固定値ではなく，入力できるようにしたものです．

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

use strict;
use warnings;

# 入力
my $r = <STDIN>;
chomp($r);

# 出力
my $pi = 3.14;
print("半径は ", 2 * $pi * $r, "\n");
```

13.5.4 4.

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

use strict;
use warnings;

```

```
# 入力
open(my $rfh, "<", "ex7_4_in.txt") or die;
my $op1 = <$rfh>;
chomp($op1);
my $op2 = <$rfh>;
chomp($op2);
close($rfh) or die;

# 出力
open(my $wfh, "<", "ex7_4_out.txt") or die;
print($wfh $op1 + $op2, "\n");
print($wfh $op1 * $op2, "\n");
print($wfh $op1 x $op2, "\n");
close($wfh) or die;
```

13.6 8. 練習問題

13.6.1 1.

まっすぐ書くとこのようになります.

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

use strict;
use warnings;

open(my $fh, "<", "ex8_1_in.txt") or die;
while (my $line = <$fh>) {
    print($line);
}
close($fh) or die;
```

ファイル名を指定できるようにすると，以下のようになります.

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

use strict;
use warnings;

```

```perl
my $filename = <STDIN>;
chomp($filename);

open(my $fh, "<", $filename) or die;
while (my $line = <$fh>) {
    print($line);
}
close($fh) or die;
```

13.6.2 2.

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

use strict;
use warnings;

my $i = 1;
while ($i <= 10) {
    print($i, "\n");
    $i = $i + 1;
}
```

13.6.3 3.

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

use strict;
use warnings;

my $i = 1;
my $sum = 0;
while ($i <= 10) {
    $sum = $sum + $i;
    $i = $i + 1;
}
print($sum, "\n");
```

13.6.4 4.

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

use strict;
use warnings;

my $n = <STDIN>;
chomp($n);

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

13.6.5 5.

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

use strict;
use warnings;

my $n = 1;
while ($n < 20) {
    print($n, "\n");
    $n = $n + 3; # 3ずつ離れている
}
```

13.6.6 6.

素直に書くとこのようになるでしょう.

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

```

```
use strict;
use warnings;

my $input = <STDIN>;
chomp($input);
while ($input ne "quit") {
    $input = <STDIN>;
}
print("end...", "\n");
```

というのを二回書きたくなかったら． スカラ変数が初期化時には，undef であることを利用して， 以下のように書くこともできます．

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

use strict;
use warnings;

my $input;
while (!defined($input) or $input ne "quit") {
    $input = <STDIN>;
}
print("end...", "\n");
```

13.7 9. 練習問題

13.7.1 1.

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

use strict;
use warnings;

my @chars = ("alpha", "bravo", "charlie", "delta", "echo");
my $i = 0;
while ($i < 5) {
    print($chars[$i], "\n");
```

```
    $i = $i + 1;
}
```

13.7.2 2.

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

use strict;
use warnings;

my $r = <STDIN>;
chomp($r);
if ($r < 0) {
    $r = 0;
}

my $pi = 3.14;
print(2 * $pi * $r, "\n");
```

13.7.3 3.

"0" は，偽なので，print() は実行されません．

13.7.4 4.

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

use strict;
use warnings;

open(my $fh, "<", "ex9_4_in.txt") or die;
my $sum = 0;
while (my $n = <$fh>) {
    chomp($n);
    $sum = $sum + $n;
```

```
}
close($fh) or die;

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

13.8 10. 練習問題

13.8.1 1.

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

use strict;
use warnings;

# 入力
my $ymd = <STDIN>;
chomp($ymd);
my ($y, $m, $d) = split(/,/, $in);

# 出力
print($y, "年", $m, "月", $d, "日です", "\n");
```

13.8.2 2.

数値を読み込んで，配列の添字に使うことがわかればできます．リストの添字は，0 から始まるので，入力する数字と 1 つずれていることに注意が必要です．

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

use strict;
use warnings;

my @names = ("fred", "betty", "dino", "wilma", "pebbles", "bamm-bamm");
while (my $n = <STDIN>) {
    chomp($n);
```

```
    print $names[$n - 1]; # 添字とズレているので注意
}
```

14 付録

14.1 perli / perl -de0

簡単に調査したいときに毎回ファイルを作って実行するのは面倒ですよね．
例えば，perli というコマンドがあります．

perli のダウンロード

```
$ curl -LO https://raw.githubusercontent.com/mklement0/perli/master/bin/perli
$ chmod +x perli
$ ./perli    # 又は，perl perli
```

perli は，Perl プログラムなので，ダウンロードしたらそのまま使うことができます．perli を実行すると，簡単に値を調べることができます．

```
> 1
1
> defined(undef)
''
> defined(10)
1
> not 1
''
> (1, 2, 3)
1
2
3
```

色々試してみてもいいかもしれません．

同じ用途で，perl -de0 というコマンドも使えますが，若干使い方が難しいです．こちらは，perl があれば最初から使えるでしょう．

```
$ perl -de0
DB<1> x 1
0  1
DB<2> x defined(undef)
```

```
0  ''
DB<3> x (1,2,3)
0  1
1  2
2  3
```

x を先頭に付けてコマンドを実行します．結果はリスト値が添字と一緒に順に縦に並びます．

15　さいごに

練習問題や文章の一部は,『初めての Perl』 のものをお借りしています．素晴しいテキストありがとうございます．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．勉強した方の生活がどこか楽になってくれると良いと思います．