Perl Entrance #3
参考資料

2012年 3月 18日

Perl入学式代表取り締まられ役 papix
(papix2011@gmail.com)

1. 前回の復習

Sample Code

```
use strict;
use warnings;
use utf8;

binmode STDIN, ":utf8";
binmode STDOUT, ":utf8";

my $age;
$age = 22;
$age = 23;
my $school = 'マクド出禁学院大学';
my $name = <STDIN>;

chomp($name);

print "私の名前は${name}です．年齢は
${age}歳です．${school}に通っていま
す.¥n";
print '私の名前は${name}です．年齢は
${age}歳です．${school}に通っていま
す.¥n';
```

```
my $x = 10;
my $y = 2;

my $num = $x + $y;
my $div = $x / 10;

print "${x} + ${y} = ${num}¥n";
print "${x} / 10 = ${div}¥n";
```

お約束

```
use strict;
use warnings;
use utf8;

binmode STDIN, ":utf8";
binmode STDOUT, ":utf8";
```

- Perlでプログラミングをする際の“お約束”.
  - 最初に必ず書くようにします.

変数と文字の入力

```
my $age;
$age = 22;
$age = 23;
my $school = 'マクド出禁学院大学';
my $name = <STDIN>;

chomp($name);
```

- 変数の名前には“$”を付けます.
- 変数は“my”をつけて宣言します(1行目).
  - 宣言しないと, 実行時に警告されます(use strict;の効果).
- 変数には値や文字列を代入することができます(2･4行目).
  - 変数の中身は一番最後に代入されたものになります(2･3行目).
- 変数の宣言と代入は同時に行えます(4行目).
- <STDIN>で, キーボードから文字列を入力します(5行目).
- chomp関数で変数の末尾の改行コードを取り除きます(6行目).

文字の出力

```
print "私の名前は${name}です．年齢は${age}歳です．${school}に通っています.¥n";
print '私の名前は${name}です．年齢は${age}歳です．${school}に通っています.¥n';
```

- print関数で文字を出力することができます.
  - 出力したい文字列を""(ダブルクォーテーション)で囲むと, その中の変数は展開されます.
    - $name = 'papix', $age = 23, $school = 'Perl入学式'なら,
      私の名前はpapixです. 年齢は23歳です. Perl入学式に通っています.
      と表示されます.
  - 出力したい文字列を''(シングルクォーテーション)で囲むと, その中の変数は展開されず, そのまま出力されます.
    - 私の名前は${name}です. 年齢は${age}です. ${school}に通っています.¥n
      と表示されます.

演算

```
my $x = 10;
my $y = 2;

my $num = $x + $y;
my $div = $x / 10;

print "${x} + ${y} = ${num}¥n";
print "${x} / 10 = ${div}¥n";
```

- よく使う演算子として, 以下のようなものがあります.
  - + (加算), - (減算), * (乗算), / (除算)
  - % (剰余算) … あまりを求める. 例: 5 % 3 = 1
  - ** (指数演算) … 左の数を右の数だけ掛け合わせる. 例: 2 ** 3 = 2 * 2 * 2 = 8
  - . (文字列結合) … 左の文字列と右の文字列を結合する
    例: ‘Perl ‘ . ‘Entrance‘ = ‘PerlEntrance‘
- 演算を“()”でくくることによって, 演算順序を変更できます.
  - 例: 2 + 4 * 3 = 2 + 12 = 14　(2 + 4) * 3 = 6 * 3 = 18
- 演算の順序やその機能は, 中学･高校の数学と同じイメージです.

Let's try!

1. [目標: 10分]
   キーボードから半径を入力して, その円の面積を求めるプログラムを書いてみよう. ただし, 円周率は3.14とします.

   半径が1なら面積は3.14, 半径が5なら78.5くらいになるはずです.

   ※円の面積は, “半径 × 半径 × 円周率”で求めることができます.

2. 条件分岐

条件分岐の例

1万円以上持っているなら → 寿司を食べる
そうでないとき, 5000円以上持っているなら→ 焼肉を食べる
そうでないとき, 財布に500円以上持っているなら→ 牛丼を食べる
そうでないなら→ 諦める

- “条件を満たすなら, … をする”や, “そうでないなら, … をする”という処理は, 条件分岐を使って書くことができます.
- 条件分岐を使えば, Perlに複雑な処理･計算を行わせることができるようになります.
- まず, 最も単純な例を考えてみます.

所持金が1円以上 → ご飯を食べる
そうでないなら → 何も食べない

- これをPerlで書いてみると, 次のようなコードになります.

Sample Code

```
use strict;
use warnings;
use utf8;

binmode STDIN, ":utf8";
binmode STDOUT, ":utf8";

print "所持金はいくら?>>>";
chomp(my $money = <STDIN>);

print "所持金は${money}円なので...¥n";
if ( $money > 0 ) {
        print "何か食べに行く.¥n";
} else {
        print "何も食べない.¥n";
}
```

条件分岐

- if文の構造は単純です.

```
if ( "条件" ) {
        条件を満たした時("条件"が真の時)の処理
} else {
        条件を満たしていない時("条件"が偽の時)の処理
}
```

- else以下は省略できます.

```
if ( "条件" ) {
        条件を満たした時("条件"が真の時)の処理
}
```

- この場合, “条件”を満たさない場合は何も行いません.
- if文で厄介なのは, この“条件”の示し方です.
  - 次のページで, “条件”の示し方について説明していきます.
  - 慣れない間は難しいかもしれませんが, 資料を参考に一歩ずつチャレンジしていきましょう!

関係演算子

- 関係演算子は2つの数値や文字列の関係を比較する演算子です.

数値用	文字列用	意味
$x == $y	$x eq $y	$xと$yが同じ
$x != $y	$x ne $y	$xと$yが異なる
$x >= $y	$x ge $y	$xは$y以上($x==$yも成立)
$x > $y	$x gt $y	$xは$yより大きい($x==$yは成立しない)
$x <= $y	$x le $y	$xは$y以下($x==$yも成立)
$x < $y	$x lt $y	$xは$yより小さい($x==$yは成立しない)
$x <=> $y		$xと$yを比較し, $x < $y なら-1, $x == $y なら0, $x > $y なら1を返す
	$x cmp $y	$xと$yを比較し, $x lt $y なら-1, $x eq $y なら0, $x gt $y なら1を返す

- 数値の比較と文字列の比較で関係演算子が異なります.
  - 文字列の比較は辞書順(aが小さく, zが大きい)で行います.
- 関係演算子は, 関係を満たすなら真を, そうでないなら偽を返します.
  - 例: $x = 1, $y = 1のとき, $x == $y … 真
    $x = 1, $y = 2のとき, $x == $y … 偽

論理演算子

- 論理演算子は, 複数の関係演算子を組み合わせる為に使います.

演算子	名前	記述例	意味
&&	論理積(AND)	x && y	xとyが共に真なら真, そうでないなら偽
||	論理和(OR)	x || y	xとyの少なくとも1方が真なら真, そうでないなら偽
!	否定(NOT)	!x	xが真なら偽, 偽なら真

- 関係演算子と論理演算子を組み合わせることで, 複雑な条件を書くことができます.
  - 例: $x = 1, $y = 'perl', のとき, ($x == 1) && ($y eq 'perl') … 真
    $x = 1, $y = 'perl', のとき, ($x == 1) && ($y eq 'Python') … 偽
    "$y eq 'Python'"は偽となる.
    $x = 1, $y = 'perl', のとき, ($x == 1) || ($y eq 'Python') … 真
    "$y eq 'Python'"は偽だが, "$x == 1"は真なので真.
- それでは, 次のような条件分岐をPerlで書いてみましょう.

1000円以上持っていて, かつ空腹度が80以上なら → ご飯を食べる
そうでないなら → 何も食べない

Sample Code

```
use strict;
use warnings;
use utf8;

binmode STDIN, ":utf8";
binmode STDOUT, ":utf8";

print "所持金はいくら?>>>";
chomp(my $money = <STDIN>);

print "空腹度はいくら?>>>";
chomp(my $hungry = <STDIN>);

print "所持金は${money}円, 空腹度は${hungry}なので...¥n";

# (財布に1000円以上) かつ (空腹度が80以上)
if ( ( $money >= 1000 ) && ( $hungry >= 80 ) ) {
        print "晩ご飯を食べる.¥n";
} else {
        print "何も食べない.¥n";
}
```

elsif

- if文を複数使用したいときは, elsifを使います.

```
if ( 条件1 ) {
        条件1を満たした時(条件1が真のとき)に行う処理
} elsif ( 条件2 ) {
        条件1を満たしていないとき，条件2を満たしている時
        (条件1が偽で，条件1が真のとき)に行う処理
} else {
        条件1も条件2も満たしていない時(条件1も条件2も偽のとき)に行う処理
}
```

  - C言語では“else if”と書きますが, Perlでは“elsif”です.
    よく間違えるので注意しましょう.
  - elsifは, 1つのif文の中で複数回使うことができます.

- それでは, elsifを使って次のような条件分岐を書いてみましょう.

もし現在地が’umeda’で, かつ2000円以上持っているなら → 居酒屋に行く
そうでないとき, 1000円以上持っているなら → 吉野家に行く
そうでないなら → 何もしない

Sample Code

```
use strict;
use warnings;
use utf8;

binmode STDIN, ":utf8";
binmode STDOUT, ":utf8";

print "所持金はいくら?>>>";
chomp(my $money = <STDIN>);

print "現在地は?>>>";
chomp(my $name = <STDIN>);

if( $name eq 'umeda' && $money >= 2000) {
        print "居酒屋に行く.¥n";
} elsif( $money >= 1000 ) {
        print "吉野家に行く.¥n";
} else {
        print "何もしない.¥n";
}
```

Let's try!

```
use strict;
use warnings;
use utf8;

binmode STDIN, ":utf8";
binmode STDOUT, ":utf8";

print "所持金はいくら?>>>";
chomp(my $money = <STDIN>);

print "現在地は?>>>";
chomp(my $name = <STDIN>);

if( $name eq 'umeda' && $money >= 2000) {
        print "居酒屋に行く.¥n";
} elsif( $money >= 1000 ) {
        print "吉野家に行く.¥n";
} else {
        print "何もしない.¥n";
}
```

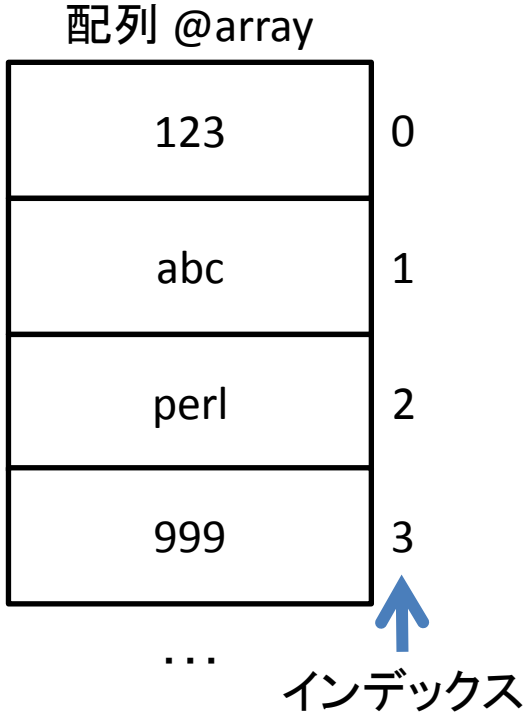
先程のSample Codeを実際に書いてみましょう.
余裕があるなら, 条件を付け加えたりしてアレンジしても構いません.
きちんと書いたはずなのに動かなかったり, 期待通りの動作をしないなら, アシスタントの人を呼んで, その原因を一緒に考えてみましょう.

目標: 10分

3. 配列とその操作

配列

- 変数には1つのデータしか格納できませんが, 配列は複数のデータを格納することができます.
  - 変数が“データを格納する箱”なら, 配列は“データを格納するタンス”です.
- 変数に格納されたデータは, 0以上の数値のインデックスを持ちます.
  - インデックスを用いることで, 変数に格納された要素にアクセスすることができます.

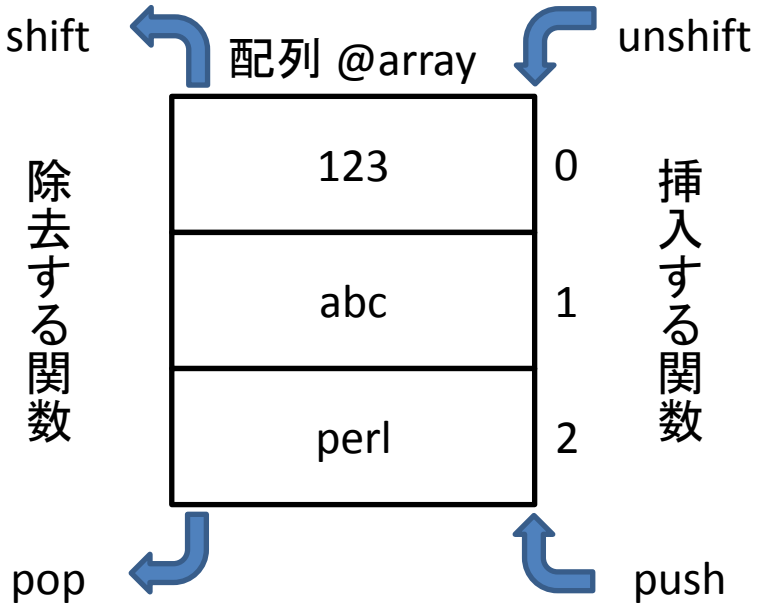


Sample Code

```
# 以降，Sample Codeでは“お約束”のコードを省略します．

# 変数の宣言と同じく，“my”が必要．
my @array;
my @array2 = ('banana', 'orange', 'apple');
# この場合，$array2[0] = 'banana', $array[1] = 'orange', $array[2] = 'apple'
となります．

my $word = 'strawberry';
$array[0] = $word;
# 配列arrayのインデックス0に，$word1のデータを代入する．
# @array[0]ではなく，$array[0]である点に注意しましょう．

$array2[3] = 'grape';
# 配列array2のインデックス3に，‘grape’を代入する．

print "${array}[0]¥n"; # "strayberry"と表示されます．
print "@{array2}¥n";
# こうすると，"banana orange apple grape"のように，
# 配列の中身が半角スペースを区切りにして表示されます．
```

配列の操作(1)

- インデックスを利用する以外に, push/popやshift/unshiftといった関数を利用して配列を操作することもできます.
  - push(@array, $word); … @arrayの末尾に$wordを挿入する.
  - pop(@array); … @arrayの末尾の要素を取り除く.
    $word = pop(@array); とすれば, $wordに取り除かれた要素が代入される.
  - unshift(@array, $word); … @arrayの先頭に$wordを挿入する.
  - shift(@array); … @arrayの先頭から要素を取り除く.
    $word = shift(@array); とすれば, $wordに取り除かれた要素が代入される.



Sample Code

```
my @array;
my @array2;

print "文字を入力してください(1回目)>>>";
chomp(my $input = <STDIN>);
push(@array, $input); # @arrayの末尾に$inputの中身(入力した文字列)を挿入する
unshift(@array2, $input); # @arrayの先頭に，$inputの中身を挿入する

print "文字を入力してください(2回目)>>>";
chomp(my $input = <STDIN>);
push(@array, $input); # @arrayの末尾に$inputの中身(入力した文字列)を挿入する
unshift(@array2, $input); # @arrayの先頭に，$inputの中身を挿入する

my $shift = shift(@array2); # @array2の先頭の要素を取り除き，$shiftに代入する

print "@{array}¥n";
# 例えば，'Perl'，'Ruby'と入力した場合，'Perl Ruby'と表示されるはずです．
print "@{array2} / ${shift}¥n";
# 例えば，'Perl'，'Ruby'と入力した場合，
# 'Ruby / Ruby'と表示されるはずです．
```

配列の操作(2)

- 配列を操作する上で便利な関数をいくつか紹介します.
  - @array = sort(@array); … @arrayに含まれる要素をソートします.
  - @array = reverse(@array) … @arrayに含まれる要素を逆順にします.
  - $element = @array; … @arrayに含まれる要素数を$elementに代入する.
  - join($split, @array);
    … @arrayに含まれる要素を, $splitで区切って文字列にします.
    - 例: @array = ('Eila', 'Sanya', 'Nipa'); print join(':', @array);
      Eila:Sanya:Nipaと表示されます.
  - split(/$split/, $str);
    … $strに含まれる文字列を$splitの文字で区切って配列にします.
    $splitの前後に"/"が必要なことに注意.
    - 例: $str = 'Eila:Sanya:Nipa'; @array = split(/:/, $str);
      @array = ('Eila', 'Sanya', 'Nipa')となります.

Sample Code

```
my @array = ('Nanoha', 'Hayate', 'Testarossa');
my @array2 = sort @array;
# @arrayの要素がソートされて@array2に代入されます.
# @array2 = ('Hayate', 'Nanoha', 'Testarossa'); になります
print "@{array}¥n";
print "@{array2}¥n";

my $num = @array; # $numに, @arrayの要素数(この場合, 3)が代入されます
print "${num}¥n";

my $str = join(',', @array);
# $strに, @arrayに含まれる要素を","を区切り記号にして代入します.
# この場合, $str = 'Nanoha,Hayate,Testarossa'となります
print "${str}¥n";

my $word = 'Eila:Sanya:Nipa';
my @array3 = reverse(split(/:/, $word));
# この場合, @array3 = ('Nipa', 'Sanya', 'Eira'); となります.
print "@{array3}¥n";
```

Let's Try!

```
my @array;
my @array2;

print "文字を入力してください(1回目)>>>";
chomp(my $input = <STDIN>);
push(@array, $input);
unshift(@array2, $input);

print "文字を入力してください(2回目)>>>";
chomp(my $input = <STDIN>);
push(@array, $input);
unshift(@array2, $input);

my $pop = shift(@array2);

print "@array¥n";
print "@array2 / $pop¥n";
```

先程のSample Codeを実際に書いてみましょう.
余裕があるなら, 条件を付け加えたりしてアレンジしても構いません.
きちんと書いたはずなのに動かなかったり, 期待通りの動作をしないなら, アシスタントの人を呼んで, その原因を一緒に考えてみましょう.

目標: 10分

Tips

- reverseは配列に使えば"配列の要素を逆順"にしますが, 変数に使えば変数に代入されている文字列を逆順に並べ替えます.

```
my $str = 'Perl';
$str = reverse $str;
print ${str};
# "lreP"と表示される.
```

4. for文/while文

for文/while文

- プログラムのある部分を複数回繰り返したい, という場合があります. そのような時に使うのが, for文とwhile文です.
- まず, for文を使ったプログラムの簡単な例を示してみます.

Sample Code

```
for my $num (1..10) {
        print "Hello, world!¥n";
}

my $max = 14;
for my $num (1..$max) {
        print "${num}¥n";
}
```

for文

- これは, "Hello, world!"という文字列を10回表示し, その次に1から$max(Sample Codeの場合, 14)までの値を出力するプログラムです.
- for文の構文は, 簡単に書けば次のようになります.

```
for ループ内で使う変数（リスト）{
        繰り返したい処理
}
```

  - ループ内で使う変数
    - ここで変数を設定することで, "繰り返す値"を"繰り返したい処理"の中で使うことができます. 必ずしも設定する必要はありません.
  - リスト
    - "(1, 2, 3)"などのように, スカラ(内部に構造を持たないデータ構造. 変数や数値, 文字列など)の集合をリストと呼びます.
    - "$x..$y"は"($x, $x+1, … , $y-1, $y)"というリストになります.
    - for文は, リストの先頭から最後までを"ループ内で使う変数"に代入しながら"繰り返したい処理"を繰り返します.

while文

- 一方, while文の構文は, 次のようになります.

```
while (条件) {
        繰り返したい処理
}
```

- while文では, "条件"が真である限り, "繰り返したい処理"を繰り返します.
  - Perlでは, 1は真となるので, 次のようなコードは無限ループとなります.

```
while (1) {
        print "Hello, world!¥n";
}
```

  - 無限ループに陥った場合は, Ctrlキーとcキーを同時押し(Macの場合は, Commandキーと.キーを同時押し)で強制終了することができます.

last

- for文やwhile文を強制的に抜けたいときには, lastを使います.

```
while (1) {
        print "Hello, world!¥n";
        last; # "Hello, world!"を1回だけ表示して，whileのループを抜ける
}
```

Sample Code

```
for my $count (1..10) {
        print "${count}回目>>>";
        chomp(my $word2 = <STDIN>);

        if($word1 eq $word2) {
                print "${count}回目で正解です!¥n";
                last;
        }
}
```

for文と配列

- for文の"リスト"には, 次のように配列を使うこともできます.

```
my @array = ('Madoka', 'Sayaka', 'Homura', 'Mami', 'Anzu');
for my $word (@array) {
        print "${word},";
        # "Madoka, Sayaka, Homura, Mami, Anzu"と表示
}
```

- 配列に用いるsort関数を使えば, こんなこともできます.

```
my @array = ('Madoka', 'Sayaka', 'Homura', 'Mami', 'Anzu');
for my $word (sort @array) {
        print "${word}¥n";
        # ソートされ, "Anzu, Homura, Madoka, Mami, Sayaka"と表示.
}
```

Sample Code

```
my @array;

while(1) { # 無限ループを発生
        chomp(my $input = <STDIN>); # 文字を入力し...
        if($input eq 'end') { # 入力した文字が'end'なら, ループを終了
                last;
        } else {
                push(@array, $input);
                # そうでないなら, @arrayの末尾に
                # $inputの中身(入力した文字列)を挿入する
        }
}

my $num = 1;
for my $word (sort @array) {
        print "${num} : ${word}¥n";
        $num++;
}
```

Let's Try!

```
my @array;

while(1) { # 無限ループを発生
        chomp(my $input = <STDIN>);
        if($input eq 'end') {
                last;
        } else {
                push(@array, $input);
        }
}

my $num = 1;
for my $word (sort @array) {
        print "${num} : ${word}¥n";
        $num++;
}
```

先程のSample Codeを実際に書いてみましょう.
余裕があるなら, 条件を付け加えたりしてアレンジしても構いません.
きちんと書いたはずなのに動かなかったり, 期待通りの動作をしないなら, アシスタントの人を呼んで, その原因を一緒に考えてみましょう.

目標: 10分

Tips

- C言語のようなfor文も使えます.

```
for(my $number = 1 ; $number <= 10 ; $number++) {
        print "${number} ¥n";
}
```

…が, あまり使うことはありません.

5. 演習問題

問題1

- [目標: 10分]
  キーボードから10個の数値を入力して, それらの合計(和)を計算するプログラムを書いてみよう.
- [OPTION]
  - 'end'と入力するまで数値の入力を続けることができるようにプログラムを書き換えてみよう.
  - 入力した値が"1, 2, 3, 4, 5, 6, 7, 8, 9, 10"ならば,
    "1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55"のように, 計算式と計算結果の両方が表示されるよう, プログラムを書き換えてみよう.

問題2

- [目標: 15分]
  変数$strに設定した文字列を, キーボードから入力して当てる"文字あてゲーム"を作ってみよう.
  - ただし, 正解した際には, 以下の情報を表示すること.
    - 文字列を入力した回数(回答した回数)
    - 正解するまでに入力した文字列の一覧
  - 例えば, $str = 'Perl'で, 入力した文字列が'Ruby', 'C', 'Perl'なら,

```
正解です！ 正解するまでに3回の入力を必要としました.
正解までに入力した文字列は，'Ruby'，'C'，'Perl'です！
```

  のようになります. 細かい文面は変更してもOKです.

- [OPTION]
  - 文字列を入力する度に, その文字列が$strと比べて辞書順で前か後ろかを表示するようにしてみよう(文字列の関係演算子を使おう!).

問題3

- [目標: 15分]
  1から36までの数値について,
  - その数が15の倍数なら, "Fizz Buzz"
  - その数が15の倍数ではないが5の倍数なら"Buzz"
  - その数が15の倍数ではないが3の倍数なら"Fizz"
  - それ以外なら, その数値

  と表示するプログラムを書いてみよう.
- ちなみにこのような問題を"FizzBuzz問題"と言います.
  - 答えはこのようになるはず…
    1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16, 17, Fizz, 19, Buzz, Fizz, 22, 23, Fizz, Buzz, 26, Fizz, 28, 29, Fizz Buzz, 31, 32, Fizz, 34, Buzz, Fizz
- [OPTION]
  - キーボードから整数を入力し, 1から入力した値までのFizzBuzz問題を解くプログラムを書いてみよう.
  - ただし, 0以下の値が入力された場合は, エラーを表示すること.