ゼロからプログラミング講座 (Perl) #1 @越谷 講義ノート

ターミナルとテキストエディタ

ターミナル(Ubuntuは”端末”と呼ぶ)と、テキストエディタを開いてもらう。

テキストエディタは、Macの場合、Sublime Text2、Ubuntuの場合はデフォルトで入ってるgedit。

以降、Perlの話よりもターミナルの使い方がメイン。

which perl

which perl

- perlがどこにあるか
- （どのPerlを使っているか)
- そもそもPerlが入っているか
- 確認できる。

/usr/bin/perl

想定される反応はこういうもの。（環境によって異なるが)

次に、ディレクトリの構成について解説

lsコマンド

- 今いるディレクトリのファイル・ディレクトリ一覧を表示するコマンド
- ls
- ls -l　パーミッションや所有者名など詳細を表示
- ls -a　「.」ではじまる隠しファイルも表示
- ls -la -lと-aをあわせたもの
- など。lsはlistの略。他にもオプションがある。

cdコマンド

- cd [行きたい場所]

change directoryの略

- cd /

とやって、一番上の階層「/」に行ってもらって、それからユーザーのデフォルトの場所まで戻ってきてもらった。

lsを使っていまいる場所にどんなディレクトリがあるか探ったり、 tabキーを打ってコマンドの補完をしたり、↑キーを使ってコマンドの履歴を呼び出したりする練習

下記のコマンドで一発で戻れる

- cd ~/

「~」このニョロのマークはチルダと読む

Hello, World を書いて保存

やっとPerlの話になる。

プログラム初心者のはじめの一歩、Hello, Worldを表示するプログラムを書く。 テキストエディタで下記のように書く。 最初は、改行文字（¥n あるいはバックスラッシュｎ）はなしで。

```
#!/usr/bin/perl
print "Hello, World";
```

そして、これを適切な場所に保存する。 今回は、Document（あるいは書類）というディレクトリに「sample.pl」という名前で保存してもらった。（この操作はGUIで行う）

そして、先ほどの cd コマンドや ls コマンドを駆使して、ターミナル上でそのファイルを見つけ出し、実行してもらう。

sample.plの実行

sample.plが置いてあるディレクトリまで移動できたら、

perl sample.pl

とやって実行してもらう。

Hello, Worldの解説

１行目はシバン行（シェバン行）と呼ばれているが、おまじないとしておく。

最初は、行末の改行文字なしで実行してもらい、改行文字がないと結果がどうなるかを実感してもらった。

それから、改行文字あり。

```
#!/usr/bin/perl
print "Hello, World¥n";
```

- Hello, World を囲っているのはダブルクォート（シングルクォートではない）
- 全部半角文字でうつ
- print と「 " 」の間にはスペースを必ず入れる

それから、ダブルクォートをシングルクォートにするとどうなるか、実験。 シングルクォートにすると、変数展開しないため、¥nは改行にならず、そのまま￥ｎ（えんまーくえぬ）と表示されるのを確認。

プログラムを実行するもうひとつの方法

実は

```
perl sample.pl
```

とやって実行する場合は、最初の一行のシバン行（#!/usr/bin/perl）はいらない。 なぜなら、実行するときに、perlと打っているから。

そして、次のようにも実行できる。

```
./sample.pl
```

しかし、現状では「Permission Denied（権限がありません）」と表示されて、実行できない。

権限を変更する方法 chmod コマンド

- 読み、書き、実行の３種類の権限の組み合わせ
- 読み = read = r, 書き= write = w, 実行 = execute = x
- user, group, otherの３種類の実行者分類
- 権限を変更するのは chmod コマンド
- change mode
- userに実行権限を付与するとき：chmod u+x sample.pl
- 権限をなくすとき：chmod u-x sample.pl
- u+xとかではなく、644とか755とか数字で表現することもできる

和暦（平成）を西暦 に変換するプログラム

Hello, Worldができたところで、変数の扱いを知るために、与えられた数字を西暦に変換するプログラムを作る。

```
#!/usr/bin/perl
$jyear = 25;
$year = $jyear + 1998;
print "$year¥n”;
```

これを実行すると、「2013」という答えが返ってくる。

意味合いとしては、入力された数字を平成の年だと解釈して、それを西暦に変換してる。

もう少し、結果をわかりやすくするために、下記のように加工。

```
#!/usr/bin/perl
$jyear = 25;
$year = $jyear + 1998;

print "平成 $year 年は $year 年です。¥n";
```

こうすると、「平成 25 年は 2013 年です。」という結果がでてくるようになる。

値を入力できるようにする（標準入力を受け取る）

いまの状態だと、平成21年が西暦何年か、平成３年が西暦何年か調べるために、いちいちプログラムを書き換えなければならない。そこで、平成何年を調べたいかプログラムに尋ねさせて、それに答えて（数字を入力して）結果を出力するという風に変更する。

```
#!/usr/bin/perl
print "平成？";
$jyear = <STDIN>;
$year = $jyear + 1998;
print "平成 $ｊ year 年は $year 年です。¥n";
```

こうすると、「平成？」と画面に出た後に入力待ちになり、そこに数字を打ち込んでエンターを押す（例えば20）と、

平成 20
年は 2008 年です。
という結果になる。

ここで、結果が二行になってしまっているのは、表記ミスではなく、実際にそうなってしまう。

この場合は、20と打った後にエンターキーを叩くので、そのエンターキーによって入力された改行がそのまま反映されてしまうからである。

その余分な改行を取り除くにはchompという関数を使う。

```
#!/usr/bin/perl

print "平成？";
$jyear = <STDIN>;
$year = $jyear + 1998;

chomp( $jyear );
print "平成 $jyear 年は $year 年です。¥n";
```

こうすると、余分な改行は出てこない。

- STDINはStandard Inputのこと。日本語だと標準入力。
- これを書くと、ターミナルの入力を受け付ける
- STDINを囲んでいるくの字カッコ（< と >）は見た目のためではなく意味がある
- STDINという文字列は予約されていて、他の用途では使えない
- STDINの他に、STDOUT, STDERRというのがある

変数とは

変数とは文字列とか数値とかを１個格納するもの。

上記のプログラムでは、$jyearと書かれているところに、例えば25と書けば、実行できるがそうすると、別の数字で実行したいときに全部書き換えなければいけない。

しかし、変数でおいておけば、その変数に値を代入する行だけ書き換えればよい。

今回のコードでは、和暦の数字を格納するための変数の変数名として「$jyear」という変数名を使った。jyearの j はJapanのJのつもりだったが、j を l（エル）とか i（アイ）とかと見間違える人が多かったし、そのあとで出てくる、「$year」という変数と１文字しか違いがなく、それも間違いのもとなので、 $jyear という名前の付け方は良くなかったようだ。

というわけで、 $jyear という変数名はやめて、 $input という変数名にしましょうと提案、実際に書き換えてもらった。

```
#!/usr/bin/perl

print "平成？";
$input = <STDIN>;
$year = $input + 1998;

chomp( $input );
print "平成 $input 年は $year 年です。¥n";
```

ポイントは、$jyearだったところで、全て、残らず、書き換えることである。１個でも漏らしたり、スペルミスをすると、動かない。（動くかもしれないが、望む挙動をしてくれない。）

Perlにおける変数名は、先頭に必ず「＄（ドルマーク）」がつくこと。これが、他の言語とは異なる特徴で、他の言語の場合、こういうマークは一切つかない。

配列、ハッシュ

変数の次は配列とハッシュ。配列は値を複数持つもの。

```
@array = ('John', 'Taro', 'Hanako');
```

このように、表現する。変数と違い、先頭の文字は「@（アットマーク）」である。

配列にアクセスするときは何番目の要素かを指定してアクセスする。

$array[0] だったら John

$array[1] だったら Taro

$array[2] だったら Hanako

ということになる。注意することは、１番目からはじまるのではなく、０番目から始まるということ。

ハッシュは、配列の時は添字（何番目の要素か）という情報をもとにアクセスしたが、それだと不便なときがある。そういうときに、キーワードになる文字列でアクセスできるようなキーと値のペアのことである。

```
%hash = ( 'Taro' => 'Japanese', 'John' => 'American');
```

この場合、Taro と Japanese という文字列がペアになっている。

要素にアクセスするときは次のようにする。

$hash{'Taro'}　とすると　Japanese　という値が出てくる

配列とハッシュで注意することは、まず、個々の値を呼ぶときは先頭の文字が@や%ではなく、変数と同じ$になっていることである。

配列：@array　配列の要素：$array[0]

ハッシュ：%hash　ハッシュの要素：$hash{'Taro'}

次に、カッコの形も重要である。配列は添字を書き込むときに、角カッコで囲む。一方、ハッシュの時は弓カッコで囲む。

配列の要素：$array[0]　← 添字 0 は [ ] で囲われてる

ハッシュの要素：$hash{'Taro'}　← キー'Taro' は { } で囲われてる

配列を使ってみる

```
@array = ('Yamada', 'Tanaka', 'MyName', 'Suzuki');
```

上のような配列を与えられた時に、自分の名前を表示するようにする。（私の名前はMyNameです）

解答

```
#!/usr/bin/perl

@array = ('Yamada', 'Tanaka', 'MyName',
'Suzuki');

print "私の名前は $array[2] です¥n";
```

ポイント

- 添字は３ではなく、２である（０番目からはじまる）
- ダブルクォートでくくる
- そのほかに、確認として、ダブルクォートをシングルクォートに変えてみたり、「"私の名前は" . $array[2] . "です¥n" 」と文字列を分割し、ピリオドを使って結合してみたりしました。