＜Perl 入門１＞

Perl とは、テキスト処理、システム管理、ネットワークプログラミング、CGI、プログラミングツールなどの広い応用範囲を持つプログラミング言語です。

最初に、準備の手順を Mac と Windows では異なるので、説明したいと思う。

Mac の場合

学校のパソコンの場合 JeditX がもともと入っているのでインストールする必要がない。
しかし、家出する場合は、ｍｉをインストールする必要があるので下記のＵＲＬからインストールしておく。
http://mimikaki.net/index.html

Windwos の場合

サクラエディタがない場合、下記の URL からインストールしておく。
http://members.at.infoseek.co.jp/sakura_editor/snapshot.html
下記の URL から Windows 版をダウンロードする。[1]
http://www.activestate.com/Products/ActivePerl/Download.html
そしてインストールを行う。
今回は、Mac での Perl の使用方法を紹介する。
Per 言語でプログラムを作成するには、

① プログラムを書く

② 文法チェックを行う

③ 実行する

Perl 言語でプログラムを書くといったとき､私たちが行うことは Perl 言語の分布に従って、スクリプトを作成することになります。Mac で行う場合、そのスクリプトは、JeditX というテキストエディタを使って作成します。そこで作成したスクリプトをターミナルというものを使い実行します。
実行するには、

[1] http://www.site-cooler.com/kwl/perl/environment.htm#download

① JeditX で Perl 文を作成する

たとえば、Hello!!と表示させたい場合は

```
print “Hello!!”;
```

と JediX に入力します。

そして、ファイル名を hello.pl で保存します

② ターミナルを開き perl hello.pl と入力

うまく実行できれば Hello!と表示されます。

Perl でプログラムを書く場合、基本的には文の最後に「;」を書きます。これを書かなければ、プログラムがエラーになってしまいます。これは覚えておきましょう。

また、Perl では、print された文は表示された後、改行をしてくれません。そこで print した文を改行させるためには print”バックスラッシュ n”を書く必要があります。

Mac でのバックスラッシュの入力の仕方は

Option キーを押しながら¥記号を入力したら表示されます。

「以降、バックスラッシュは¥と表記する。」

かならず、”¥n”を書く際はこのように「””」ではさんでください

例：

```
print “Hello!!”;
print “¥n”;
```

こうすることで、実行した際に Hello!!が表示され、しっかりと改行がされる。

Perl 文では、文字以外のものも表示できる。

たとえば、１＋１＝２という計算をする場合、+は記号の一種である。

これを print 文にする場合、

```
print 1 + 1;
```

と入力する。するとターミナルで実行すれば、答えである２が表示される。

使用できる算術演算子は以下のもの。+、-は半角記号で入力。×、÷、は以下の記号を使う。

加算…足し算(+)

減算…引き算 (-)

乗算…掛け算(*)　アスタリスク

除算…割り算(/)　スラッシュ

表示：

```
print ((2 * 2) + (3 /1));
```

このように括弧を使えば、複数の演算を組み合わせることもできる。これをターミナルで実行すれば、７と表示されます。

それでは以上のことを参考に、問題を解いてみよう。

計算結果の代入

計算するだけではなく、もう少しわかりやすい説明で表そう。問題＜１＞の「１から１０まで足すと 55 です」というような表示をしたいときは、

```
my $gokei = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 ;
print $gokei;
```

この$gokei が変数になるので、そこに計算結果が代入される。

そして、print $gokei; と書くことで、$gokei が実行されたときに表示される。

ここで二つの文を見比べてください。最初の文では変数$gokei の前に my がついていますが、次の文では my はかかれていません。Perl では新しく変数を書く場合、その変数が最初に出てきたときだけ変数の前に my をつけます。

また、

```
print “1 から 10 まで足した結果は”. $gokei .“です。”
```

すると、「1 から 10 まで足した結果は 55 です。」と表示される。

ここで、上のプログラムを見てください。”” で括弧っている部分と、$gokei の部分の間に.(ドット)が入ったいます。Perl では文章の連結をする際は、ドットでつなげる。そうすることで、表示した際に、文字が連結して表示される。

変数を使った計算

結果だけではなく、個々の数値も変数に代入してから計算することができる。

：表示例＜４＞の問題を使って

計算式の個々の項目も変数を使って書くことができる。

```
my $tate = 4;
my $yoko = 7;
my $menseki = $tate * $yoko;
print “縦” . $tate . “cm”. “横” . $yoko . “cm の長方形の面積は” . $menseki . “cm2 である。”;
```

問題

＜１＞1 から 10 までの数を足してみよう。

＜２＞1 個 125 円のりんごを 3 個、1 個 98 円のみかんを 15 個買って、消費税を外税でプラスすると、総額いくらか。

＜３＞縦 4cm、横 7cm の長方形の面積を求めよ。

＜４＞半径 13cm の円の面積を求めよ。

＜５＞上底が 4cm、下底が 7cm、高さが 5cm の台形の面積を求めよ。

＜Perl 入門２＞

文字列とは

たとえば、「大谷大学」というように、文字の並んだテキストを指す。

文字列をプログラムの中で使うときには、テキストを二重引用符（”)、で囲む。

二重引用符で始めた文字列は、必ず二重引用で閉じる。

例：

```
print “大谷大学”；
```

このように書く。

文字列を変数に代入する

例：

```
my $name = ″Yoichi fukuda″;
my $mail = ″yfukuda  ¥@tibet.que.ne.jp″;
print $name . ″ From:″ . $mail;
print ″¥n″;
```

このように文字列（“yoichi Fukuda ”）を変数$name に代入しとくと、変数の値を変えるだけで、print 文のところはそのままに、表示の内容を変えることができる。これはもっと長いプログラムを作るときには是非とも必要なことである。プログラムの最初の部分に、そのプログラムの中で使う文字列をこうして変数に代入しておくと見やすくなる。

また三行目のプログラムを見てください。print $name . ″From:″ . $mail;と、$name, “From”, $mail と文字列の間には.(ドット)が書かれています。

Perl では文字列の連結は+ではなく、.(ドット)を使用します。

この様に計算した結果を再び同じ変数に代入することもできる。

例：

```
my $tat e = 5;
my $yoko =7;
print $tate * $yoko;
```

また、上記の様に、print 文の内容に計算式を直接書いてもいい。次にユーザーが入力したものを変数に代入するということを学びましょう。

では、以下の文章を実行してみよう。

```
print “名前を入力してください。”;
my $name= <STDIN>;
chomp($name);
print $name . “さん。こんにちは”;
print “¥n”;
```

一行目で「名前を入力してください。」と表示させる文で、二行目の my $name =<STDIN>;という文は、「ユーザーが入力した 1 行の文字列を変数$name に代入する」ものです。また chomp($name);という文は「変数$name の最後の改行を取り除く」ものです。<STDIN>を使って入力した 1 行の最後には改行が付加されているので、文字列の比較を行う前に取り除いておきます。

問題

<1>

いろいろな質問をし、その結果を適当な言葉を補いながら表示をする。

続いて円の面積の計算を学んでもらう。

円の面積の公式を作る場合は以下のようになる。

```
my $hankei = 5;
my $menseki = $hankei * $hankei * 3.14;
print ″半径″. $hankei . ″cm″. ″ 円の面積は″.
int($menseki) . ″です。¥n″;
```

ここで最後の文に使われている int についてだが、

int(文字列)…浮動小数点数の整数部分のみを取り出す(切捨てする)場合に用いる。

これは覚えておきましょう。

問題

<2>

では半径をユーザーに入力してもらい、円の面積を求めるプログラムを書いてみよう。

＜Perl 入門３＞

配列

ここでは、配列について学んでもらう。ここで、使うのは**@ARGV** である。

これは、コマンドライン引数といい、コマンドライン引数とは コマンドラインから、 Perl スクリプトに配列の要素を渡すことが出来る仕組みの事である。この仕組みを使えばスクリプトの動作をコマンドラインから渡した値によって変更するように汎用的にすることも出来る。

例：

```
print join(", ", @ARGV);
print "¥n";
```

ここで初めて出てきた **join** を使うと、配列の要素を連結して１つの文字列にすることができます。

これを実行するときにはいつもとは違った実行をします。仮に、上のプログラムの名前を、argv.pl とする。この argv.pl を通常

```
perl argv.pl
```

と実行するのだが、この配列の時には

```
perl argv.pl otani kyousan bukkyou
```

とかく、そうすると

otani, kyousan, bukkyou

と表示される。

これは、プログラムの１行目の print join(", ", @ARGV);　という部分で、"" で囲まれている(,)で実行したときの otani kyousan bukkyou　という文字を(,)で連結しているのである。

これは、[連結された文字列] = join [区切り文字列], [配列] ;

ということである。

＜Perl 入門４＞

処理の場合分け

今までのプログラミングでは、プログラムは一行ずつ順に実行され、一行の処理が終わったら、次の行の処理が実行された。

次は、条件によって処理をいくつかの場合に分けるやり方を導入しよう。これは日本語で考えると、ごく単純なことであるのが分かる。

もし、A が B であったら、

こうしよう。

そうでなければ、

ああしよう。

このような「もしも」の場合は、日常生活でもよくあることだが、プログラムでも、この場合分けを必要とする。

そのため Perl では、この「もし、～ならば､」や「そうでなければ､」の部分を英語で表現する。

```
if （条件式）{
        条件が真の場合に実行する処理(A)
} else {
        条件が偽の場合に実行する処理(B)
}
```

という形になっているわけです。条件が真のとき、(A)が実行され、条件は偽のとき(B)が実行されます。ここで、条件がどうであろうとも、

・(A)と(B)のどちらか一方は必ず実行される。

・(A)と(B)の両方が実行されることはない。

ということを覚えていてください。2 つの道(A)と(B)のどちらか一方のみをかならず通るのです。

具体的な例

ユーザーにパスワードを入力してもらい、

もし、そのパスワードが予め登録されているものと同じ**ならば**

「ログインできました」と表示する。

そうでなければ、（パスワードが一致しなければ）

「パスワードが違います」と表示する。

以下のプログラムを実行してみよう。

```
print ″パスワードを入力してください。″;
print ″¥n″;
my $line = <STDIN>;
chomp ($line);
if ($line eq ″1234″) {
        print ″ログインできました。″;
        print ″¥n″;
} else {
        print ″パスワードが違います。″;
        print ″¥n″;
}
```

正しいパスワードかどうかの判断は、

$line eq ″1234″

という条件式で行っています。変数$line の内容が”1234”に文字列として等しいかどうかを調べています。

ここで使われている eq は両辺を文字列として比較し、等しかったら真になるという演算子です。数の比較を行う＝＝と似た機能を持ちます。

複数の場合分け

上の説明では、条件は、イエスの場合とノーの場合の二者択一である。

しかし、場合によっては、「1 の場合、2 の場合、3 の場合、その他の場合」のように**複数の場合分け**が必要になることもある。

その場合、二番目以降の条件節は **elsif 〜:**を使う。

（１の場合）

if A が 1 である:

これこれをする。

（２の場合）

elsif A が 2 である：

**　　ああする。**

（3 の場合）

elsif A が 3 である：

**　　こうする。**

（それ以外、そうでなければ）

else：

**　　そうする。**

以下のようにすると、改行された通りに表示される。

```
print ″どの面積を求めますか？″;
print ″¥n″;

print ″1 =  三角形″;
print ″¥n″;

print ″2 =  正方形″;
print ″¥n″;

print ″3 =  長方形″;
print ″¥n″;

print ″4 =  終了″;
print ″¥n″;
my $keisan = <STDIN>;
if ($keisan==1) {    ＃一つ目の場合
        print ″三角形の面積を求めます。″;
        ＃ここに底辺と高さを聞いて面積の計算をして、結果を表示するプログラムを書く。
} elsif ($keisan==2) {
        print ″正方形の面積を求めます。″;
```

```
        ＃ここに一辺の長さを聞いて、面積の計算をして、結果を処理するプログラムを書く。以下同様
} elsif ($keisan==3) {
        print "長方形の面積を求めます。";
        } else {
        print "終了します。";
        print "¥n";
}
```

問題<1>

上の例で省略されている処理の部分を補い、プログラムを完成させて、動かしてみよう。

ファイル名は menseki.pl にする。

問題<2>

三角形、正方形、長方形以外の面積を求められるように項目を増やしてみよう。(たとえば、円の面積や台形の面積など)

＜Perl 入門５＞

ある一群の処理を、ある条件のもとで繰り返す構文を勉強しよう。

まず、もっとも基本となる構文は、

○○の条件が成り立っている間：

条件式。

繰り返し実行する部分

これを Perl では、

while(○○の条件が成り立っている間)条件式:

```
while（条件式）{
        繰り返し実行する部分
}
```

と書く。Perl では、条件式が数と文字列で変わる。

数値	文字列	意味
$x == $y	$x eq $y	等しい （equal to）
$x != $y	$x ne $y	等しくない(not equal to)
$x < $y	$x lt $y	$x は$y より小さい(less than)
$x > $y	$x gt $y	$x は$y より大きい(greater than)
$x <= $y	$x le $y	$x は$y 以下(less than or equal to)
$x >= $y	$x ge $y	$x は$y 以上(greater than or equal to)
$x <=> $y	$x cmp $y	$x と$y が等しいなら 0 $x が$y より小さいなら-1 $x が$y より大きいなら 1 compare

では、以下の問題を解いてみよう。ファイル名は **while.pl** とする。

上のプログラムを応用して、1 から 10 までの数を合計するプログラムを書いてみよう。数を足していくための変数が一つ必要になる。

```
my $i = 0;
my $gokei = 0;

while ($i < 10 ) {
        $i = $i + 1;
        $gokei = $gokei + $i;
}
print $gokei;
print "¥n";
```

まず、変数を２つ用意する。まず$i と$gokei の変数に 0 を代入する。

そして while 文　while ($i < 10)　この条件は「$i が１０になるまで」という条件で、１０になるまで実行され続ける。

そして$i = $i + 1; では、$i に 1 を足し続ける処理であり、

次に$gokei = $gokei + $i;　は、変数$gokei に$i を代入する。

そして print $gokei; で表示させる。

問題<1>

ある数からある数までの合計を計算するプログラムに変更してみよう。

つまり、最初の出発点と最後の数とをそれぞれ変数に代入して、「繰り返し」をする。

繰り返しの別のパターン

```
print "暗証番号を入力してください。";
my $password = 12345;
my $line = <S TDIN>;
while ($line ne "") {
        if ($line == $password) {
                print "ログインしました。";
                last;
        } elsif ($line != $password ) {
                print "パスワードが違います。";
        }
        $line = <STDIN>;
}
```

この形式は、ログインしたときに入力を終了するときの決まった構文である。

$line = <STDIN>;を最初と最後の文で二回使っていることに注意すること。あとは処理の内容を考える。

この繰り返し処理をしている中で、ある条件が成り立ったときに、繰り返しを終了するには、last;というコマンドを使う。

繰り返しのなかで last というコマンドに出会うと、一番内側の繰り返しを終えて、次の行に実行を進める。このプログラムでは、その後に実行すべき行がないので、プログラム自体が終了する。

問題<2>

数当てゲームを作ってみよう。

一桁の数を入力してもらう。その数が、秘密の数と一致していたら、「上がり」と表示して終了する。

もし、秘密の数よりも大きければ、「大きすぎ」と表示して、再度入力をしてもらう。

逆に小さければ、「小さすぎ」と表示して、再度入力をしてもらう。

秘密の数は、プログラムの最初で変数に代入しておく。

※上にある、パスワードを入力するプログラムを応用したもの。

BMI 値とは？

BMI 値とは、ボディ・マス指数（体格指数）で、肥満度を数値化したもの。最近話題のメタボリック症候群の判定に用いられる。

もちろん、やせ過ぎについても分かる。

計算式は、

体重（kg）/（身長（m）＊ 身長（m））

つまり、体重をメートル単位の身長の二乗で割ったもの。　身長 175cm の場合は、1.75 に直して計算する必要がある

この値が 18.5 未満の場合は、痩せすぎ、18.5 から 25 未満までは適正、25 以上は肥満ということになる。

この BMI 値を求めるプログラムをいくつか作ってみよう。

問題<3>

起動するとユーザーに名前と、身長と、体重を聞き、計算して「あなたの BMI 値は○○です。」と表示するプログラムを作って下さい。

ここでも<STDIN>;を使います。身長を m(メートル)に直すのを忘れないこと。また、もし BMI 値が 18.5 以下だったら「痩せすぎ」、18.5 以上 25 未満なら「適正」、25 以上なら「肥満」という判定結果も表示するようにしてみよう。

＜Perl 入門６＞

九九の表

最初から九九の表をつくるのは、難しい。まず、一行だけを表示してみよう。つまり

1×1　1×2　1×3　.... 1×8　1×9

を計算して表示する。

ここで、かける方の値を i にして、それが 1 から 9 まで変化する。

1×i　**←ここの数字が変化する**

ここでは計算式で書いたが、実際には計算結果だけを表示する。print では、一度に一つの計算をした値を一つ表示する。それを 9 回繰り返すのである。

```
my $a = 1;
my $i = 0;
my $seki = 0;
while ($i < 9) {
        $i += 1;
        $seki = $a * $i;
        print $seki;
}
```

まず 1×i なので$a には１、$i には０を代入し、結果を表示するための変数$seki に０を代入しておく。

そして while 文の条件式には$i が 9 になるまでと書き、次の$i += 1;で９になるまで$i に１を足し続ける。そして、$seki = $a * $b　と書き print すれば１の段の九九が表示できるのである。

問題<1>

九九の表を作成しなさい。

次に、完全な九九の表を作ってみよう。プログラムを実行した際に

```
   1  2  3  4  5  6  7  8  9
-----------------------------------
1| 1  2  3  4  5  6  7  8  9
2| 2  4  6  8 10 12 14 16 18
3| 3  6  9 12 15 18 21 24 27
4| 4  8 12 16 20 24 28 32 36
5| 5 10 15 20 25 30 35 40 45
6| 6 12 18 24 30 36 42 48 54
7| 7 14 21 28 35 42 49 56 63
8| 8 16 24 32 40 48 56 64 72
9| 9 18 27 36 45 54 63 72 81
```

上記のように表示させたい。

こうするためには、上のプログラムを 9 回繰り返すのではなく、×の前の数値も繰り返しで処理をする。つまり、繰り返しが二重に出てくる。

かけ算は **a * b** であり、ある一行は、**b** が 1 から 9 まで繰り返す。それで一つの while 文ができる。さらにその while 文が a を 1 から 9 まで繰り返すもう一つの while 文が来る。

まず、それぞれという$a,$b という変数に 1 を代入する。

```
my $a =1;
my $b =1;
```

そして、計算の答えを代入するための変数$seki も準備する。

```
my $seki =0;
$seki = $a * $b;
```

そしてここから while 文となる。

ここで新たに **pritnf** という関数を使う。この printf というのは表示桁をそろえるのに使う。使い方としては

```
printf “%3d”, $a;
```

こうすることで、$a が三桁分あけて右寄せに表示されるので、表などを表示させたいときに有効である。

今回は、１から九九の表を表示させるのは難しいので、下記に記したプログラムの中で、太字の部分だけを書き換え、九九の表を作成する。

```
my $a =1;
my $b =1;
my $seki =0;
$seki = $a * $b;
print "  ";
while ($a が９より下の場合) {
        $a を三桁分空けて右寄せに表示;
        $a に１を足し続ける;
}
print "¥n";
print "----------------------------";
print "¥n";
$a =1;
while ($a が９より下の場合) {
        |（縦棒）を三桁分空けて右寄せに表示;
        while ($b が９より下の場合) {
                $seki =$a * $b;
                printf "%3d", $seki;
                $b に１を足し続ける;
}
        print"¥n";
        $b =1;
        $a に１を足し続ける;
}
```

＜Perl 入門 7 ＞

HTML の要素を作る関数

具体例を見てもらおう。

p(″これは、Python の関数で作った段落です。″)

と関数を呼び出すと、

** これは Python の関数で作った段落です。 **

という文字列を返す関数を作る。これを、たとえば、

print p(″ これは、Python の関数で作った段落です。″)

と print 文で表示すると、上の文字列が書き出される。あるいは、文字列の足し算で、

```
my $all = p(″   段落 1 ″) . p(″ 段落 2 ″) ;
$all .= ol( li(″   箇条書き 1″) .
        li(″箇条書き 2″) .
        li(″箇条書き 3″));
$all .= p(″ 段落 3″);
```

というように、文字列変数に全部の内容を足していって、最後にそれを print することもできる。

HTML の要素ごとに、上記のような関数を作っていく。形式は、ほとんど一緒で、タグによって囲まれる文字列を引数にとり、それをタグで囲んだ文字列を返す。

（※html タグについて分からない人は、http://www.tohoho-web.com/ を参照。この先もタグは必要になるので、ブックマークしておくことをすすめる）

例外は、**a 要素**や **img 要素**の単独のタグと呼ばれるものだが、**src** などの属性が必要な要素もある。

これは引数をとるタイプである。

特に **a 要素**は、**src** と **a** タグで囲まれるテキストとがあるので、引数を二つとる。引数が複数ある場合は、@_で渡されるのだが、引数がひとつしかない場合、関数 shift を使うと、@_は

省略される。

サンプルとして、p の場合の定義を示しておこう。

```
sub p {
        my $s = shift;
        my $p = "<p>$s</p>     ¥n";
        return $p;
}
```

my $s = shift;

のように書けば、引数が$s に受け取らせることができる。

また、a タグの場合の定義は、

```
sub a {
        my ($url, $s) =@_;
        my $a = "<a href=     ¥"$url¥">$s</a>";
        return $a;
}
```

my ($url, $s) =@_;

のように@__で引数を順番に$url,$s に受け取らせる。

要素には、段落を書き出すブロック要素と、段落の途中の一部分のみを囲むインライン要素がある。後者は a 要素や img 要素、em 要素、strong 要素などである。

これらインライン要素は、最後に改行を付けないで返す。ブロック要素は、上の p 要素の関数定義のように、最後に改行を付け加える。

HTML 文書全体に関わる要素

特別な要素に html 要素、head 要素、body 要素がある。

これらは HTML 文書全体に関わる。これも文字列を引数にとり、タグで囲んだ文字列を返す関数として定義してもいいが、そうすると、中身全部が出来上がらないと書き出せない。

そこで、これらは、

・最初の<html>から<body>までの head()関数

・最後の</body>と</html>のみ end()関数

この二つで対処する。これらは HTML を書き出す最初と最後に print するようにする。 head()関数は、title 要素の内容を引数にとる。他の head 要素の内容は、いつも同じもので構わない。

html モジュール

以上の関数定義を **html.pl** というモジュールに保存しておく。これを別のプログラムから使うようにする。そのプログラムでは、文の最初に

```
require 'html.pl' ;
```

とすることで、html.pl のほかのファイルの内容を読み取らせることができる。

以下のタグを処理する関数を作る。

a, blockquote, br, dd, dl, dt, em, h1, h2, h3, head,

hr, html, img, li, ol, p, pre, strong, ul

モジュールができたら、実際に使用してみよう。

問題<1>

このプリントの上の方にある項目を参考に sample.pl というファイルを作り、そこに html モジュールを読み込み、以下のように表示してみよう。

```
<head>
<title>html.pl のテスト</title>
</head>
<body>
<p>段落 1.</p>
<p>段落 2.</p>
<ol>
<li>箇条書き 1
<li>箇条書き 2
<li>箇条書き 3
</ol>
<p>段落 3.</p>
</body>
</html>
```

問題<2>

上記の応用として、人文情報学科サイトのトップページの簡略版を作ってみよう。段組の必要はないので、全てテキストでよい。