たかし君と関数

たかし君は毎朝、学校に行く前にコンビニでカレーパンとメロンパンを買います。
カレーパンは 150 円、メロンパンは 100 円です。

今日はお昼ごはんの分として 1 つずつ買いました。
パン代は 150 円 *1+100 円 *1 で 250 円になります。

次の日は部活があるので、カレーパンをひとつ多く買いました。
パン代は 150 円 *2+100 円 *1 で 400 円になります。

さらに次の日、今日は部活がある上、親が夜に外出するので晩御飯の分もあわせて、
カレーパン 3 個、メロンパン 2 個を買いました。
パン代は 150 円 *3+100 円 *2 で 650 円になります。

たかし君は毎日同じ様な計算をするのが嫌になってしまい、楽にパン代を計算できないかと考えました。

< 考えてみよう >
何度も同じことをしているのはどこで、毎回違うのはどんなところでしょうか？

毎回同じような計算を書くのは大変なので、それぞれのパンの個数を与えると合計金額を計算する関数を作ってみましょう。

※注意！ 実際には PHP では関数名や変数名には英数字と '_' しか使えません

PHP では function というキーワードの後に、関数名 (引数のリスト){ 関数の内容 } という形で関数を定義します。

```
function パン代 ($ カレーパンの個数 ,$ メロンパンの個数 ) {
        $ 代金合計 = $ カレーパンの個数 * 150 + $ メロンパンの個数 * 100;
        return $ 代金合計 ;
}
```

上の例では、" パン代 " が関数名、"$ カレーパンの個数 " と "$ メロンパンの個数 " が引数、"$ 代金合計 " が返り値となります。

定義した関数はそれだけでは実行されません。関数を実行するためには、プログラム中の関数を実行したい場所で関数名 (引数に渡す値のリスト) と記述します。

引数は関数の内部 ("{" と "}" の間) で、関数の中だけで使用できる変数となります。
関数を使用する (関数を呼び出す) と、引数にはそれぞれ関数を使用するときに指定された値が代入されます。(指定された順番に、引数に代入されます)

返り値とは、関数を実行した結果の値のことです。1+1= のような計算式を計算すると 2 という答えが得られますが、関数と返り値の関係は計算式と答えの関係に似ています。

メロンパンの値段が変わった場合を考えてみましょう
(100 円から 120 円に値上がりしました)。

関数を使わない場合

たかし君は毎朝、学校に行く前にコンビニでカレーパンとメロンパンを買います。
カレーパンは 150 円、メロンパンは 120 円です。

今日はお昼ごはんの分として 1 つずつ買いました。
パン代は 150 円 *1+120 円 *1 で 270 円になります。

次の日は部活があるので、カレーパンをひとつ多く買いました。
パン代は 150 円 *2+100 円 *1 で 400 円になります。

さらに次の日、今日は部活がある上、親が外出するので晩御飯の分もあわせて、
カレーパン 3 個、メロンパン 2 個を買いました。
パン代は 150 円 *3+120 円 *2 で 690 円になります。

上の例で、一箇所修正モレが発生しているのをぱっと見つけられたでしょうか？
本来は 3 箇所修正する必要があるのですが、2 箇所しか変更していません。
この例では修正する場所が近いのでまだ分かり易いのですが、修正する場所が複数のファイルにまたがる場合も多く、モレなく修正することが大変な場合も多いです。

関数を使う場合

たかし君は毎朝、学校に行く前にコンビニでカレーパンとメロンパンを買います。
カレーパンは 150 円、メロンパンは 120 円です。

関数を定義します。

```
function パン代 ($ カレーパンの個数 ,$ メロンパンの個数 ) {
        $ 代金合計 = $ カレーパンの個数 * 150 + $ メロンパンの個数 * 120;
        return $ 代金合計 ;
}
```

今日はお昼ごはんの分として 1 つずつ買いました。
パン代 (1,1) で返り値は 270(円) になります。

次の日は部活があるので、カレーパンをひとつ多く買いました。
パン代 (2,1) で返り値は 420(円) になります。

さらに次の日、今日は部活がある上、親が外出するので晩御飯の分もあわせて、
カレーパン 3 個、メロンパン 2 個を買いました。
パン代 (3,2) で返り値は 690(円) になります。

このように、関数としてまとめておくと修正は一箇所ですみます。

関数は実行すると内部のプログラムを順に処理していきますが、return 文があるとそこで関数の実行を終了して return の引数 (上の例では $ 代金合計) の値を関数の返り値として、関数を呼び出したプログラムに渡します。return の引数が無い場合や、return が無い場合は関数の返り値は NULL になります。

実際に PHP で関数の定義を書く場合は以下のようになります。

```
function price_sum($curry_num,$melon_num) {
        $sum = $curry_num * 150 + $melon_num * 100;
        return $sum;
}
```

それでは、先ほどのたかし君の買い物を関数を使って書き直してみましょう。

```
たかし君は毎朝、学校に行く前にコンビニでカレーパンとメロンパンを買います。
カレーパンは 150 円、メロンパンは 100 円です。

関数を定義します。
function パン代 ($ カレーパンの個数 ,$ メロンパンの個数 ) {
        $ 代金合計 = $ カレーパンの個数 * 150 + $ メロンパンの個数 * 100;
        return $ 代金合計 ;
}

今日はお昼ごはんの分として 1 つずつ買いました。
パン代 (1,1) で返り値は 250( 円 ) になります。

次の日は部活があるので、カレーパンをひとつ多く買いました。
パン代 (2,1) で返り値は 400( 円 ) になります。

さらに次の日、今日は部活がある上、親が外出するので晩御飯の分もあわせて、
カレーパン 3 個、メロンパン 2 個を買いました。
パン代 (3,2) で返り値は 650( 円 ) になります。
```

何度も使われる計算や処理を関数にすると、そのプログラムを関数に一度書くだけで何回でも使い回すことが出来ます。プログラムを書く量も減らせますし、またプログラムに修正があった場合でも、関数を一箇所直すだけですむので修正もれなどが発生しにくくなります。

たかし君は毎朝、学校に行く前にコンビニでカレーパンとメロンパンを買います。
カレーパンは 150 円、メロンパンは 120 円です。

パンの値段を変数に代入します。
$ カレーパンの値段 = 150;
$ メロンパンの値段 = 120;

関数を定義します。
```
function パン代 ($ カレーパンの個数 ,$ メロンパンの個数 ) {
        global $ カレーパンの値段 , メロンパンの値段 ;
        $ 代金合計 = $ カレーパンの個数 * $ カレーパンの値段 + $ メロンパンの個数 * $ メロンパンの値段 ;
        return $ 代金合計 ;
}

function 夜食代 ($ カレーパンの個数 ) {
        global $ カレーパンの値段 ;
        $ 代金合計 = $ カレーパンの個数 * $ カレーパンの値段 ;
        return $ 代金合計 ;
}
```

今日はお昼ごはんの分として 1 つずつ買いました。
パン代 (1,1) で返り値は 270(円) になります。

次の日は部活があるので、カレーパンをひとつ多く買いました。
パン代 (2,1) で返り値は 420(円) になります。

さらに次の日、今日は部活がある上、親が外出するので晩御飯の分もあわせて、
カレーパン 3 個、メロンパン 2 個を買いました。
パン代 (2,2)+ 夜食代 (1) で返り値は 690(円) になります。

global 指定することで、関数の外で代入された値を使うことが出来るようになりました。
条件分岐などを使用して、複数の return を切り替えて異なる返り値を返すことも出来ます。

関数にするメリットをまとめてみます。
1) 同じことを何度も書かなくてすむ
2) 修正するときに一箇所だけで良い
3) ひとまとまりの処理に名前を付けられる
4) 安全に変数を使える

4 番目のメリットですが、関数の中で使う変数は関数の外、または他の関数の中の変数とは、たとえ変数名が一緒でも、違うものとして扱われます。
関数の外側、または同じ関数の中では、同じ変数名の変数は当然同じものとして扱われますので、離れた場所で複数使われていると、意図せず同じ変数名を使ってしまって、変数の値の上書きをしてしまうことがあります。
そのような場合でも、関数にすることで変数を誤って上書きをするミスを減らすことができます。

これまでの例ではそれぞれのパンの価格は固定でしたが、しばしば価格が変わり、他の関数でも使用する場合にはどうしたら良いでしょうか。

夜食代を計算する関数を作成して、さらにカレーパンの値段と、メロンパンの値段を変数にして共有します。

```
たかし君は毎朝、学校に行く前にコンビニでカレーパンとメロンパンを買います。
カレーパンは 150 円、メロンパンは 120 円です。

パンの値段を変数に代入します。
$ カレーパンの値段 = 150;
$ メロンパンの値段 = 120;

関数を定義します。
function パン代 ($ カレーパンの個数 ,$ メロンパンの個数 ) {
        $ 代金合計 = $ カレーパンの個数 * $ カレーパンの値段 + $ メロンパンの個数 * $ メロンパンの値段 ;
        return $ 代金合計 ;
}

function 夜食代 ($ カレーパンの個数 ) {
        $ 代金合計 = $ カレーパンの個数 * $ カレーパンの値段 ;
        return $ 代金合計 ;
}

今日はお昼ごはんの分として 1 つずつ買いました。
パン代 (1,1) で返り値は 0( 円 ) になります。

次の日は部活があるので、カレーパンをひとつ多く買いました。
パン代 (2,1) で返り値は 0( 円 ) になります。

さらに次の日、今日は部活がある上、親が外出するので晩御飯の分もあわせて、
カレーパン 3 個、メロンパン 2 個を買いました。
パン代 (2,2)+ 夜食代 (1) で返り値は 0( 円 ) になります。
```

上のような関数では、値段が全て 0(円) になってしまいます。

関数の中と関数の外では同じ名前の変数でも別物として扱われるため、関数の中の変数である、$ カレーパンの値段、$ メロンパンの値段は、一度も代入されていない変数、つまり値は NULL になります。
NULL は計算上は 0 として扱われますので、パンの個数がいくつであっても、代金合計は 0 になります。

関数の外と変数を共有するためには、global と言う構文を使用します。関数の外側にある変数のことをグローバル変数、関数の中の変数のことをローカル変数と呼びます。
global $ 変数名 1,$ 変数名 2; と関数の中で書くと、グローバル変数の $ 変数名 1、$ 変数名 2 をローカル変数のように扱うことが出来ます。$_POST,$_GET のような変数は、global 指定しなくても関数の中で使える特殊な変数でスーパーグローバル変数と呼ばれます。