函数

函数基础

函数的定义：

形式：

function 函数名 (【$形参 1】 【，$形参 2】 【，.... 】){

//函数体。。。。。。

}

说明：

1，定义时使用的形参，其实就是一个变量——只能在该函数内部使用的变量

2，形参作为变量，其名字是“自己定义”——自然应该遵循命名规范；

函数的调用：

函数名（$实参 1， $实参 2， .....）；

说明：

1，实参应该跟要调用的函数的形参“一一对应”；

2，实参就是“数据值”，可能是直接值（比如 5，”abc”），也可能是变量值(比如$v1)

函数调用详细过程

1，首先，将函数调用时的实参数据，传递（赋值）给函数的形参（变量）；

2，程序的执行流程，进入到函数内部——此时可以认为是一个跟外界“隔离” 的“独立运行空间”。

3，在函数内部，按正常的流程顺序，执行其中的代码；

4，直到函数结束，则退出该运行空，而返回到原来调用函数的位置，继续执行后续代码！

5，如果在函数内部执行的过程中，有 return 语句，则也会立即终止函数，并回到函数调用位置。

函数的参数问题

函数形参的默认值问题

我们可以给一个函数定义时的形参，赋值一个“默认值”，则这个函数调用的时候，该形参对应的实参，可以不给值。

函数形参的默认值，可以只给部分形参设置默认值，但设置默认值性的形参，都要放在“右边”（后边）：

形参的传值问题

一句话：形参的传值问题，其实就是“变量之间的传值问题”：

其实无非就是实参变量，传值给形参变量的问题。

即：

此时，也同样有两种传值方式：

值传递：

这是默认值。如果没有特别设定，参数传值都是值传递。

引用传递：

需要在形参的前面加 &符号

函数参数的数量问题

1，通常，函数调用时的实参数量，应该跟函数定义时的形参数量保持一致。

2，如果函数定义时，形参有默认值，则对应的实参就可以进行一定程度的省略：

注意：省略只能从右往左进行。

3，有一种定义和使用函数的特别形式（并不常见）：它不定义形参，而实参任意给出。

其实，系统中也有类似的函数：，比如：

```
var_dump($v1);
var_dump($v1, $v2, $v3);     //ok!
```

可见，该函数就可以接受任意个数的实参；

我们自己也可以定义这种函数。其实，这种函数，依赖的是以下 3 个系统函数来获取相应的信息，以得到实参数据的处理：

1: func_get_args(); //获取实参数据列表，成为一个数组

2: func_get_arg($i);　　//获取第$i 个实参数据，$i 从 0 开始算起；

3：func_num_args();　//获取实参的数量（个数）

函数的返回值问题

一个观念问题：

函数的返回值，不是语法规定，而是应用所需：需要就返回，不需要就无需返回。

返回值，一定是通过 return 语句！

形式：

```
function  函数名(....)
{
    //。。。。。。
return    XX 数据;
```

}

注意：

return 语句的作用，不管后面跟不跟数据值，都会立即终止函数的执行，返回到函数调用的位置并继续后续工作。

函数的其他形式：

可变函数

先想想可变变量：
$v1 =”abc”;
$abc = 123;
echo　$$v1;　//输出 123，这就是所谓的可变变量。
可变变量：一个变量的名字还是一个变量！
可变函数：一个函数的名字是一个变量！
演示可变函数的一个灵活性使用：

匿名函数

匿名函数就是没有名字的函数。

有 2 种形式的匿名函数：

形式 1：将一个匿名函数“赋值”给一个变量——此时该变量就代表该匿名函数了！
形式 2：

是直接将一个匿名函数，当做“实参”来使用！——即调用“别的函数 A”的时候，使用一个匿名函数来当做实参。自然，在该函数 A 中，也就应该对该匿名函数当做一个函数来用！