

## 如果在局部作用域使用（访问）全局变量？（常见需求）

有 2 种做法：

做法 1：

使用 global 关键字来实现：

```
43 echo "<hr />";
44 $v4 = 4;
45 function f4(){
46     //在函数中，使用global来声明一个要使用的全局变量的同名局部变量。
47     global $v4; //这里，$v4是局部变量，只是跟全局的v4同名
48     //实际情况是：此时外部v4变量跟内部的v4变量，共同
49     //指向一个数据区—引用关系！
50     echo "<br />在局部访问全局变量v4 = $v4";
51     $v4 = 44; //修改其值；
52 } //如果这里unset($v4); 不影响外部$v4的使用
53 f4();
54 echo "<br />在全局再次访问v4 = $v4";
55
56 ?>
57 </body>
```

全局变量\$v4

局部变量\$v4

数据: 4 -> 44

在局部访问全局变量v4 = 4  
在全局再次访问v4 = 44

做法 2：

使用\$GLOBALS 超全局变量来实现：

```
57 echo "<hr />";
58 $v5 = 5; //全局变量
59 function f5(){
60     // $GLOBALS可以认为是全局变量的另一种使用形式！
61     echo "<br />在局部访问全局变量v5 = " . $GLOBALS['v5'];
62     $GLOBALS['v5'] = 55; //修改其值；
63 }
64 f5();
65 echo "<br />在全局再次访问v5 = " . $v5;
66 echo "<br />在全局再次访问v5 = " . $GLOBALS['v5'];
67
68 ?>
69 </body>
70 </html>
```

在局部访问全局变量v5 = 5  
在全局再次访问v5 = 55  
在全局再次访问v5 = 55

但，如果我们对\$GLOBALS 变量的某个单元（也即下标）进行 unset，则其就会完全对应销毁该变量。

这是因为，\$GLOBALS 对全局变量的使用可以看做是全局变量的另一种语法形式而已，而不是“引用关系”，举例如下：

```
71 $v6 = 6; //全局变量
72 function f6(){
73     //GLOBALS可以认为是全局变量的另一种使用形式!
74     echo "<br />在局部访问全局变量v6 = " . $GLOBALS['v6'];
75     $GLOBALS['v6'] = 66; //修改其值;
76     unset($GLOBALS['v6']);
77 }
78 f6();
79 echo "<br />在全局再次访问v6 = " . $v6;
80 ?>
81 </body>
82 </html>
```

此时就是实实在在地销毁了该全局变量

在局部访问全局变量v6 = 6  
Notice: Undefined variable: v6 in  
在全局再次访问v6 =

## 有关函数的系统函数：

- `function_exists()`：判断一个函数是否被定义过。其中使用的参数为“函数名”：

```
14 if( function_exists("f1") == false ){
15     function f1(){
16         echo "<br />这个函数我自己定义了!";
17     }
18 }
19 f1(); //这里就可以放心使用该函数了!
```

- `func_get_arg($i)`：获取第  $i$  个实参值
- `func_get_args()`：获取所有实参（结果是一个数组）
- `func_num_args()`：获取所有实参的个数。

## 其他系统函数：

自己会查，并需要去查：

- [字符串函数](#)：

- 输出与格式化 : echo , print, printf, print\_r, var\_dump.
- 字符串去除与填充 : trim, ltrim, rtrim, str\_pad
- 字符串连接与分割 : implode, join , explode, str\_split
- 字符串截取 : substr, strchr, strrchr,
- 字符串替换 : str\_replace, substr\_replace
- 字符串长度与位置 : strlen, strpos, strrpos,
- 字符转换 : strtolower, strtoupper, lcfirst, ucfirst, ucwords
- 特殊字符处理 : nl2br, addslashes, htmlspecialchars, htmlspecialchars\_decode,
- [时间函数 :](#)
  - time, microtime, mktime, date, idate, strtotime, date\_add, date\_diff, date\_default\_timezone\_set, date\_default\_timezone\_get
- [数学函数 :](#)
  - max, min, round, ceil, floor, abs, sqrt, pow, round, rand

# 有关函数的编程思想

## 递归思想——递归函数

递归函数，就是：在一个函数内部调用它自己的函数！

先考察一个最简单的函数：

```
function f1($n){  
    echo $n;  
    $n++;  
    f1($n);  
}
```

f1(1);

从这个简单的函数可以看出，该函数调用是“永无止境”的（没完没了），最终会将内存消耗完毕。

显然，这不是一个正常的做法！

实用的递归函数是：能够控制这个调用的过程中，会在某个时刻（条件下）停下来！

实例演示：

求 5 的阶乘。

数学上，有这样两个有关阶乘的基本规则：

1, n 的阶乘，是 n-1 的阶乘，乘以 n 的结果。

2, 1 的阶乘是 1；

现在，假设，有一个函数，该函数“能够”计算 n 的阶乘。

```
function jiecheng($n){  
    //.....  
}
```

\$v1 = jiecheng(8); //结果应该是 8 的阶乘

\$v2 = jiecheng(5); //结果应该是 5 的阶乘

```

14 //现在, 假设, 有一个函数, 该函数“能够”计算n的阶乘。
15 function jiecheng( $n ){
16     echo "<br />开始: 有人要求{$n}的阶乘";
17     if( $n == 1){
18         echo "<br />结束: 终于求到了{$n}的阶乘: : 1";
19         return 1;
20     }
21     $jieguo = $n * jiecheng($n-1);
22     echo "<br />结束: 终于求到了{$n}的阶乘: : $jieguo";
23     return $jieguo;
24 }
25 $v2= jiecheng(5); //结果应该是5的阶乘
26 /*
27 演示调用过程:
28 $v2 = jiecheng(5)相当于:
29 $v2 = 5 * jiecheng(4)==>>
30 $v2 = 5 * (4 * jiecheng(3) ) ==>>
31 $v2 = 5 * (4 * (3 * jiecheng(2) ) ) ==>>
32 $v2 = 5 * (4 * (3 * (2 * jiecheng(1) ) ) ) ==>>
33 $v2 = 5 * (4 * (3 * (2 * 1 ) ) ) ==>>
34 $v2 = 5 * (4 * (3 * 2 ) ) ==>>
35 $v2 = 5 * (4 * 6 ) ==>>
36 $v2 = 5 * 24 ==>>
37 $v2 = 120
38 */
39 echo "<br />v2 = $v2";
40

```



递归思想总结:

当面对一个“大问题”，该大问题可以经由该问题的同类问题的“小一级问题”而经过简单计算获得，而且，可以获知（已知）这类问题的“最小一级问题”的答案。则，此时就可以使用递归方法来解决该问题。

则此时该函数的基本模式是:

```

function digui( $n){
    if(是最小一级){
        return 已知的答案;
    }
    $jieguo = 对 digui($n-1) 进行简单运算;
    return $jieguo;
}

```

课间练习:

以下数列: 1, 1, 2, 3, 5, 8, 13, .....

说明:

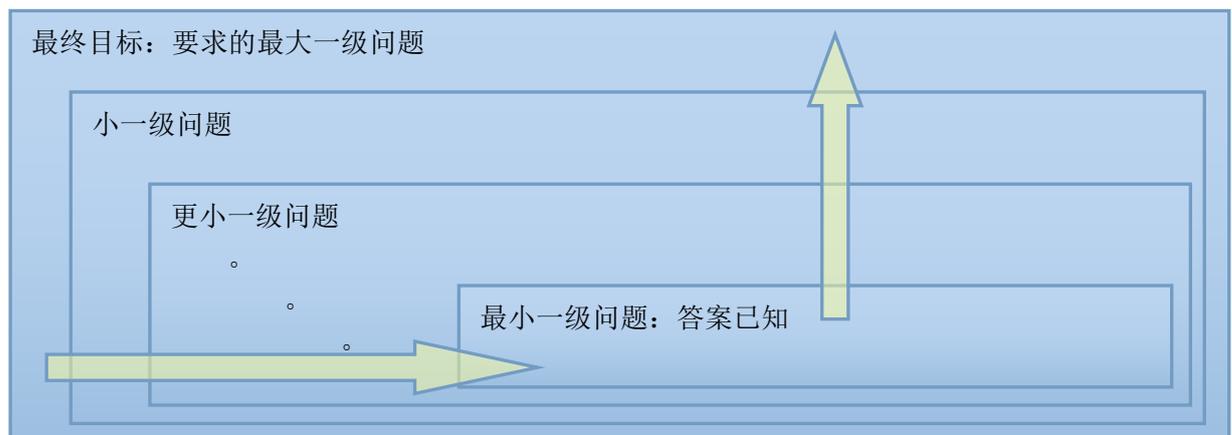
第 1 项是 1, 第 2 项也是 1 (都是已知);

其他项, 是其前两项的和;

求: 第 20 项;

```
function shulie($n){ //把 n 理解地第几项;
  if ( $n== 1  ||  $n == 2 ){
    return 1;
  }
  $jieguo = shulie($n-2) + shulie($n-1);
  return $jieguo;
}
$v1 = shulie( 20) ;
```

递归思想图示：



## 递推（迭代）思想

也同样思考这个问题：

求 5 的阶乘：

先演示最初级的做法：

```
▶ 14 //演示递推思想：
15 //目标：要求5的阶乘：
16 $jiecheng1 = 1; //表示1的阶乘（已知）；
17 $jiecheng2 = $jiecheng1 * 2; //2的阶乘；
18 $jiecheng3 = $jiecheng2 * 3; //3的阶乘；
19 $jiecheng4 = $jiecheng3 * 4; //4的阶乘；
20 $jiecheng5 = $jiecheng4 * 5; //5的阶乘；
21 ?>
```

将上述代码，使用一个变量，也同样能完成：

```

14 //演示递推思想:
15 //目标: 要求5的阶乘:
16 $jiecheng = 1; //表示1的阶乘(已知);
17 $jiecheng = $jiecheng * 2; //2的阶乘;
18 $jiecheng = $jiecheng * 3; //3的阶乘;
19 $jiecheng = $jiecheng * 4; //4的阶乘;
20 $jiecheng = $jiecheng * 5; //5的阶乘;
21 ?>

```

然后, 将上述代码的规律性体现出来——就是使用循环:

```

16 $jiecheng = 1; //表示1的阶乘(已知);
17 for($i = 2; $i <= 5; ++$i){
18     //此循环会从2的阶乘开始, 一次次求得
19     //“更大”一个数的阶乘, 直到5的阶乘
20     $jiecheng = $jiecheng * $i; //i的阶乘;
21 }

```

然后, 将该语句, 再次进行转换, 使用递推思想中的 2 个观念: 前一个答案, 后一个答案:

```

16 $qian = 1; //表示“前一个已知的答案”: 这里是第一个, 就是1的阶乘
17 for($i = 2; $i <= 5; ++$i){ //意图求得从2开始到5的“每一个数阶乘”
18     //此循环会从2的阶乘开始, 一次次求得
19     //“更大”一个数的阶乘, 直到5的阶乘
20     $jieguo = $qian * $i; //要求的结果是由“前一个结果”经过简单乘法运算而得到
21     echo "<br />{$i}的阶乘是$jieguo";
22     $qian = $jieguo; //将当前求得的“结果”, 又当成“前一个”, 以供下一次使用!
23 }
24 echo "<br />结果为: " . $jieguo;

```

递推总结:

如果要求一个“大问题”, 且该问题有如下 2 个特点:

1, 已知该问题的同类问题的最小问题的答案。

2, 如果知道这种问题的小一级问题的答案, 就可以轻松求得其“大一级”问题的答案, 并且此问题的级次有一定的规律;

则此时就可以使用递推思想来解决该问题, 代码模式为:

\$qian = 已知的最小一级问题的答案;

for(\$i = 最小一级的下一级; \$i <= 最大一级的级次; ++\$i) {

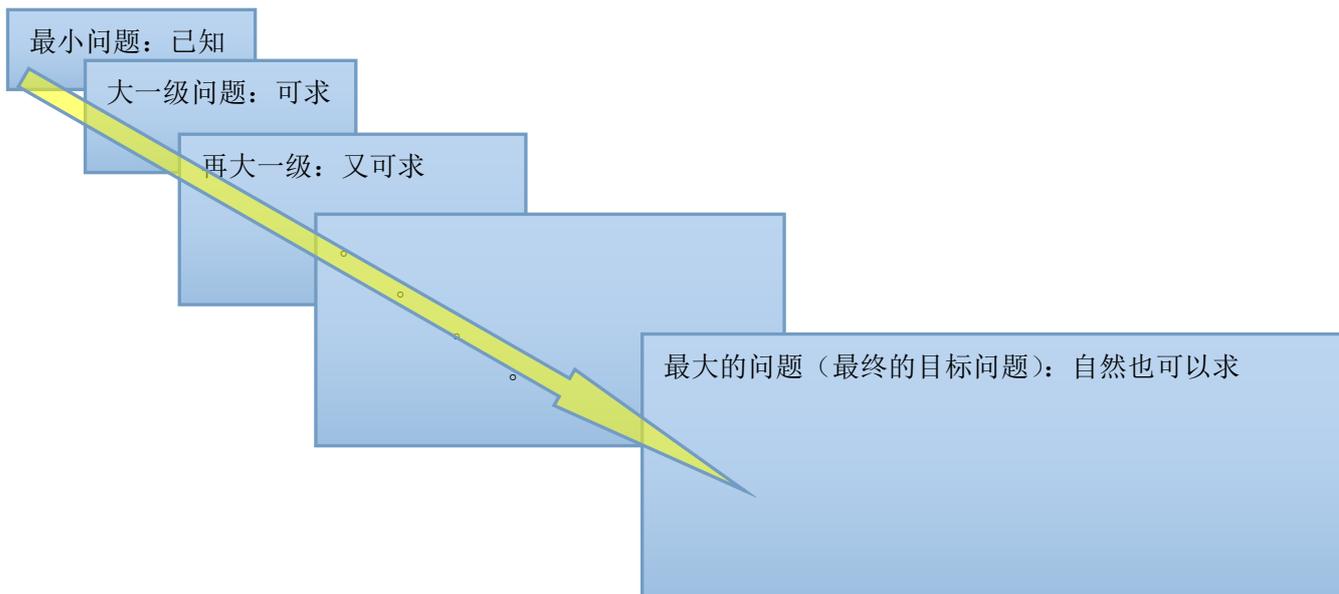
    \$jieguo = 对 \$qian 进行一定的计算, 通常需要使用到\$i;

    \$qian = \$jieguo;

}

echo “结果为: ” . \$jieguo;

递推思想图示:



通常，如果一个问题，既能使用递归算计解决，又能使用递推算法解决，则应该使用递推算  
法。

下面用递推思想来完成刚才的数列题：

以下数列：1， 1， 2， 3， 5， 8， 13， .....

求第 20 项：

```
26  /*
27  下面用递推思想来完成刚才的数列题：
28  以下数列：1， 1， 2， 3， 5， 8， 13， .....
29  求第20项：
30  */
31  $qian1 = 1;
32  $qian2 = 1;
33  for($i = 3; $i <= 20; ++$i){
34  日   $jieguo = $qian1 + $qian2; //第3项
35     $qian1 = $qian2;
36     $qian2 = $jieguo;
37  }
38  echo "<br />数列的第20项为：" . $jieguo;
39
```