

PHP 基础

目录:

1. PHP 基本语法规则.....	6
1.1. PHP 标记.....	6
1.2. PHP 语句结束符.....	6
1.3. PHP 注释.....	7
1.3.1. 单行注释:	7
1.3.2. 多行注释:	7
2. 变量.....	7
2.1. 含义与定义形式.....	7
2.2. 变量的命名规则.....	9
2.2.1. 基本规则（明规则）	9
2.2.2. 行业规则（潜规则）	9
2.3. 变量的 4 种操作.....	9
2.3.1. 赋值——常见操作.....	10
2.3.2. 取值——常见操作.....	10
2.3.3. 判断变量 isset().....	10
2.3.4. 删除/销毁变量 unset().....	10
2.4. 变量传值（难点）	11
2.4.1. 含义:	11
2.4.2. 值传递:	11
2.4.3. 引用传递:	12
2.5. 预定义变量.....	13
2.5.1. \$_GET 变量.....	14
2.5.2. \$_POST.....	14
2.5.3. \$_REQUEST.....	15
2.5.4. \$_SERVER 变量.....	17
2.6. 可变变量.....	19
3. 常量.....	19
3.1. 常量的含义.....	19
3.2. 常量的两种定义形式.....	20
3.2.1. define()函数形式:	20
3.2.2. const 关键字定义:	20
3.3. 常量的两种取值形式.....	20
3.3.1. 直接使用:	20
3.3.2. 使用 constant()函数以取值:	20
3.4. 变量与常量的区别:	21
3.5. 判断一个常量是否存在: defined();.....	21
3.6. 预定义常量.....	22



3.7. 几个魔术常量.....	23
4. 数据类型.....	25
4.1. 数据类型分类与概述.....	25
4.1.1. 标量类型:	25
4.1.2. 复合类型:	25
4.1.3. 特殊类型:	25
4.2. 整型 integer, int.....	26
4.2.1. 四种书写形式.....	26
4.2.2. 进制的相互转换.....	27
4.3. 浮点型 float.....	28
4.4. 布尔型 Boolean.....	29
4.5. 字符串型 String.....	30
4.6. 数组类型 Array.....	30
4.7. 空类型 NULL.....	31
4.8. 类型判断.....	32
4.9. 类型转换.....	33
4.9.1. 自动转换.....	33
4.9.2. 强制转换:	34
5. 运算符详解.....	34
5.1. 概述:	34
5.1.1. 含义.....	34
5.1.2. 按参与运算的数据的个数来分类.....	34
5.1.3. 按功能分类.....	34
5.2. 赋值运算符.....	35
5.3. 算术运算符.....	35
5.4. 连接运算符(.).....	36
5.5. 自赋值运算符.....	36
5.6. 自操作(自加自减) 运算符.....	37
5.7. 比较运算符.....	38
5.8. 逻辑运算符.....	40
5.8.1. 逻辑与 (&&) :	40
5.8.2. 逻辑或 () :	41
5.8.3. 逻辑非 (!) :	43
5.8.4. 逻辑运算的短路规则:	43
5.9. 条件运算符.....	45
5.10. 位运算符 (了解)	46
5.10.1. 位运算基本运算规则:	46
5.10.2. 整数的按位与 (&) 运算.....	47
5.10.3. 按位左移运算.....	47
5.11. 错误抑制符@.....	48
5.12. 运算符的优先级.....	48
6. 流程控制.....	49
6.1. 流程控制概述.....	49
6.1.1. 三大流程结构:	49



6.1.2. 流程图常用图形符号：	50
6.2. if 语句	50
6.2.1. 形式 1：单分支	50
6.2.2. 形式 2：双分支	51
6.2.3. 形式 3：多分枝	52
6.2.4. if 语句综合形式	54
6.3. 分支结构之 switch 分支语句（重点）	55
6.4. 循环结构之 while 循环语句	57
6.5. 循环结构之 do while 循环语句	59
6.6. 循环结构之 for 循环语句（重点/难点）	61
6.7. 多重循环及案例	64
6.8. 循环的中断	68
7. 函数	68
7.1. 函数的概念与作用	68
7.2. 函数的定义与调用	69
7.3. 函数执行原理（重点/难点）	69
7.4. 函数参数（重点）	70
7.4.1. 形参（形式参数）	70
7.4.2. 实参（实际参数）	70
7.4.3. 函数参数的传值方式	70
7.4.4. 形参的默认值	71
7.5. 函数返回值（重点）	71
7.6. 可变函数	72
7.7. 匿名函数	72
7.8. 系统常用函数介绍	73
7.8.1. 跟函数有关的函数	73
7.8.2. 字符串有关常用函数	73
7.8.3. 常用数学函数（重点）	74
7.8.4. 常用时间函数	74
8. 函数相关	75
8.1. 变量的作用域问题（重点）	75
8.1.1. 局部作用域与局部变量：	75
8.1.2. 静态变量：一个特殊的局部变量	75
8.1.3. 全局作用域与全局变量：	76
8.1.4. 超全局作用域与超全局变量：	76
8.2. 递归函数（重点/难点）	77
9. 文件加载	79
9.1. 文件加载的含义	79
9.2. 文件加载的四种方式（重点）	79
9.3. 四种方式的区别	80
10. 错误处理	80
10.1. 错误分类	80
10.2. 常见错误代号（重点）	81
10.3. 错误触发	82



10.4. 错误显示设置.....	82
10.5. 错误日志设置.....	83
10.6. 自定义错误处理（重点/难点）	83
11. 字符串详解.....	85
11.1. 4 种不同形式的字符串.....	85
11.1.1. 单引号字符串.....	85
11.1.2. 双引号字符串.....	85
11.1.3. heredoc 字符串.....	86
11.1.4. nowdoc 字符串.....	86
11.2. 转义字符.....	87
11.3. 字符串的长度问题.....	87
11.4. 常用字符串函数（重点）	88
12. 数组详解.....	90
12.1. 数组的概念和定义.....	90
12.2. 数组下标问题.....	90
12.2.1. 下标的可用值.....	90
12.2.2. 整数下标的特性.....	91
12.2.3. 索引数组.....	91
12.2.4. 关联数组.....	91
12.3. PHP 数组的维数.....	91
12.4. 数组的遍历（重点）	92
12.4.1. 使用 foreach 语句遍历数组.....	92
12.4.2. 使用 for 循环语句遍历数组.....	93
12.5. 常用数组函数.....	93
12.6. 数组排序算法（重点/难点）	94
12.6.1. 数组的排序问题.....	94
12.6.2. 冒泡排序算法.....	94
12.6.3. 选择排序算法.....	95
12.7. 数组查找算法.....	95
12.7.1. 二分查找：	96

昨日回顾

浏览器出发（请求）》找 hosts 文件》找网络上 DNS 服务》进入相应的服务器》Apache》PHP》mysql 数据库模块》mysql 数据库。

Apache 中调用 PHP 的设置：

```
LoadModule php7_module "php 目录/php7apache2_4.dll"
```

```
AddType Application/x-httpd-php .php .php4 .php3
```

hosts 文件中的内容：

```
127.0.0.1 www.a.com
```

```
127.0.0.1 www.b.com
```

```
12.35.121.77 www.c.com
```



127.0.0.1 www.quanzhan7.com

127.0.0.1 www.php69.com

PHP 的基本设置：

设置 php.ini 的位置：

在 apache 的主配置文件(httpd.conf)中： PHPIniDir “php 目录”

设置 php 的运行时区：

在 php.ini 中设置： date.timezone = PRC

PHP 的模块设置（以 mysqli 模块为例）

#设置 PHP 的模块所在位置（目录）

extension_dir = “php 目录/ext/”

#开启所需要的模块：

extension = php_mysqli.dll

mysql 的安装与配置：

PHP 基础课程总体内容：

环境搭建

PHP 运行环境与原理

Apache 的安装、配置与管理

PHP 的安装与配置

MySQL 的安装与配置

多站点虚拟主机配置

数据表达

PHP 的基本语法规则

变量

常量

数据类型

数据计算

运算符详解

机器码介绍

流程控制

分支结构

循环结构

代码重用：

函数

文件加载

其他：

错误处理

字符串详解

1. PHP 基本语法规则

1.1. PHP 标记

PHP 语言，是一种可以嵌入到“html”代码中的后台处理语言（程序）
有以下几种标记形式，只推荐第一种。

1, `<?php php 代码写在这里..... ?>`

这是推荐写法！

2, `<script language="php"> php 代码写在这里..... </script>`

3, `<? php 代码写在这里..... ?>`

需要到 php.ini 中进行配置：short_open_tag = On //默认为 Off，表示不能用该形式。

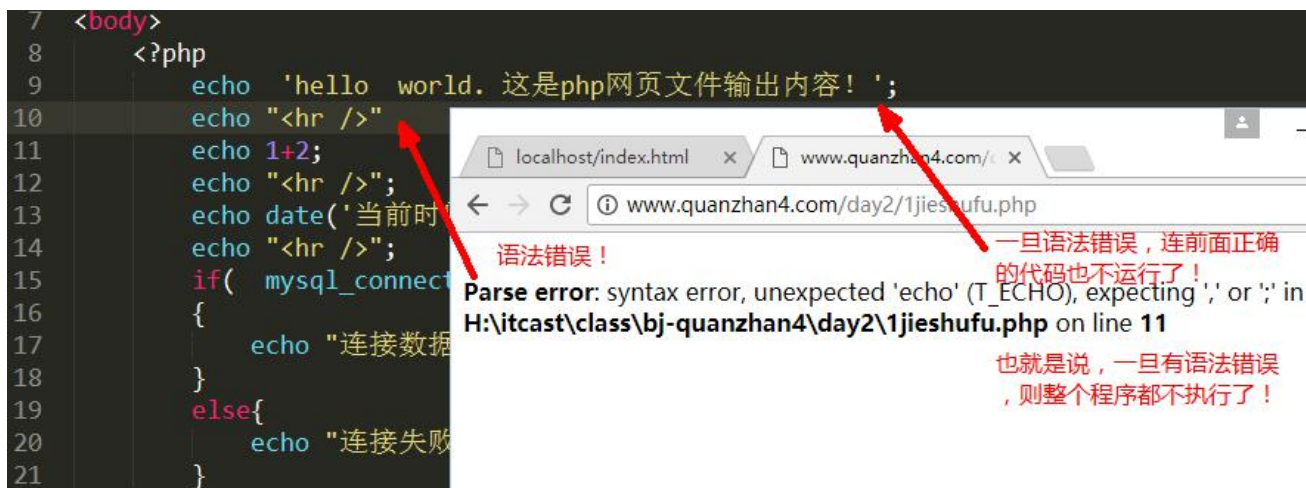
注意：

纯 PHP 代码：可以省略标记结尾符。

后面，我们会写很多“纯 php 代码文件”（里面没有任何 html 代码）。

1.2. PHP 语句结束符

使用英文分号（;）表示一条语句的结束。



1.3. PHP 注释

1.3.1. 单行注释：

两个斜杠：//斜杠后面的部分就是注释内容，PHP 语言不会去处理它（不执行）

1.3.2. 多行注释：

形式：

```
/*
```

这是注释内容。。。

可以写多行

```
*/
```

/* 当然写一行也是可以的 */

重要的事情是：注释是写程序的非常重要的因素和习惯！

2. 变量

2.1. 含义与定义形式

就是使用一个“标记符号”（标识符），来代表某个数据。

类比：

用一个名字（姓名），来代表某个人。

用一个身份证号码，来代表某个人。。。

用一个变量，就可以理解为“使用一个数据”。

既然是变量，就是该数据时可以改变的一种数据。

对应常量：就是一种不可以（不允许）改变的数据。

定义形式： **\$变量名 = 具体的数据；**

特别注意：变量名区分大小写！！！！！！

举例：

`$v1 = 123;`

//这里的等于号(=)，不是数学上的等于(相等)，而是“赋值”的意思。

//其本质含义是：将 123 这个数据，放到 v1 这个变量中去。

```
1 <?php
2 $v1 = 1;
3 $v2 = 2;
4 $v3 = $v1 + $v2;
5
6 echo $v3;
7
8 echo "<hr>";
9 echo $v1, $v2, $V3;
10
```

写成了大写
所以报错！

注意：未定义的变量 V3

Notice: Undefined variable: V3 in H:\itcast\class\bj-php-69\php_base\phpbase-day1\code\02bianliang-base.php on line 9

变量的另一个角度理解：

变量是一个“容器”，是一个“盒子”，是一个可以存放数据的位置——内存空间。

也可以这样去理解：

内存（条）中分割出很多很多的小格子，每个格子都有一个编号地址（所谓内存地址）。

其中存储了数据的格子上有一定的标记名（变量名），并存放了对应的数据。

图示如下：



0xA01	\$a	200
0xA02	\$b	100

对应代码如下：

```
$a = 200;
```

```
$b = 100;
```

其中，“0xA01”，“0xA02”代表内存地址。内存地址其实是我们看不到的，由系统分配。

2.2. 变量的命名规则

2.2.1. 基本规则（明规则）

就是给一个变量命名的时候，要遵循的规则。

- 1，构成：字母、数字、下划线；
- 2，开头：字母或下划线；
- 3，注意：变量名不要跟系统中的“关键字”（即语法所用单词）重复——关键字不多，也就几十个。

后续，我们还会学到另外一些语言或场合下有关命名的规则，其规则，跟上述几乎差不多！

2.2.2. 行业规则（潜规则）

通常，尽量将变量命名为“见名知意”。

尽量用更容易理解的单词（或拼音）去表示一个数据，很多时候可以使用“多个单词（拼音）”。

比如：

ChildAge, YungerAge, ParentHouse, MyParentHouse, MyParentHousePrice

此时，通常有两个行业中的惯例规则：

骆驼命名法（小驼峰命名法）：第一个单词首字母小写，其余单词首字母大写。

childAge, youngerAge, parentHouse, myParentHouse, myParentHousePrice

帕斯卡命名法（大驼峰命名法）：所有单词都首字母大写。

ChildAge, YungerAge, ParentHouse, MyParentHouse, MyParentHousePrice

2.3. 变量的 4 种基本操作

任何一个变量，有且只有 4 种对变量的“操作”：



2.3.1. 赋值——常见操作

将一个数据（值）放入一个变量中。

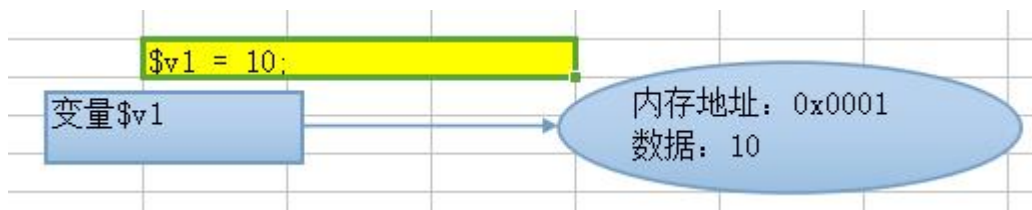
```
$name = “张三丰”;
```

几乎所有变量要想后续正常使用，第一件事就是“赋值”。

变量初始赋值后，可以后续再去重新赋值——这就是修改了变量的值，比如：

```
$name = “张三不疯”;
```

变量赋值的本质是：将变量名，跟一个数据“建立关联（联系）”，图示如下：



此时，对该变量的操作，也就是对该内存地址上的数据的操作。

2.3.2. 取值——常见操作

取值，就是，从变量这个容器中拿到其中存储的数据（值）；

取值无处不在：凡是使用到变量，并在该位置上，需要一个“数据”的时候，就会发生取值操作

2.3.3. 判断变量 isset()

就是判断一个“变量名”是否里面存储了数据！

判断的结果是：true（真，表示有），或者 false（假，表示没有）。

使用这个语法来判断：isset(\$变量名);

还有一个特殊的赋值，赋值后，变量中也没有数据，如下：

```
$v5 = null; //null 是一个特殊的“数据”（值），该数据的含义是：没有数据。
```

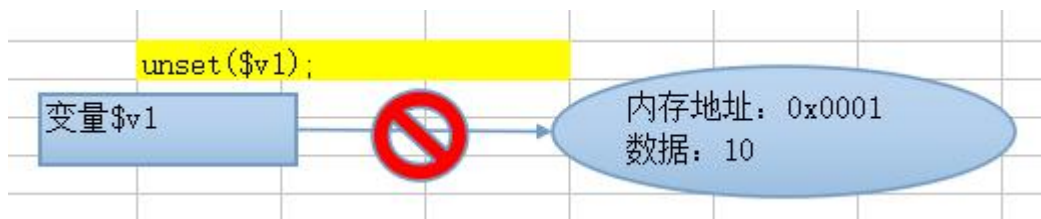
即此时判断 isset(\$v5)的结果是“false”。

2.3.4. 删除/销毁变量 unset()

当一个变量中存储了数据，我们也可以去销毁（删除）它，语法如下：

```
unset( $变量名 )。
```

删除变量的本质是：断开变量名跟其关联过的那个数据之间的“联系”，图示如下：



此时，该变量就不再指向某个数据了，其 `isset()` 判断的结果为 `false`。

```

01php-yufa.php x 02bianliang-base.php x 03bianliang-isset-unset.php x
1 <?php
2 $v1 = 1;
3 $result = isset($v1); //将对变量v1的判断结果放到$result中
4 echo $result; //实际结果是“true”，但echo输出后为“1”
5 echo "<br>";
6 var_dump($result); //这也是输出！
7 //var_dump()可以输出一个变量的完整信息
8 $result2 = isset($v3);
9 echo "<br>v3的结果为: ";
10 var_dump($result2);
11 //下面演示unset掉一个变量后的结果：
12 unset($v1); //销毁该变量！
13 $result3 = isset($v1);
14 echo "<br>v1被unset()之后为: ";
15 var_dump($result3);
16 echo $v1;
17
18
19 ?>

```

1
bool(true)
v3的结果为: bool(false)
v1被unset()之后为: bool(false)
Notice: Undefined variable: v1 in H:\itcast\class\bj-169\php_base\phpbase-day1\code\03bianliang-isset.php on line 16

2.4. 变量传值（难点）

2.4.1. 含义：

是指，将一个变量的值“传递”给另一个变量的方式问题。

形式上，就是一个变量在等号左边，一个变量在等号右边，就表示将右边变量的值传给左边的变量。

有且只有两种传数据的方式：值传递，和引用传递。

2.4.2. 值传递：

就是将右边变量的“数据值”本身，复制一份，然后赋值给左边变量。

形式如下：

`$变量1 = $变量2;`

举例：

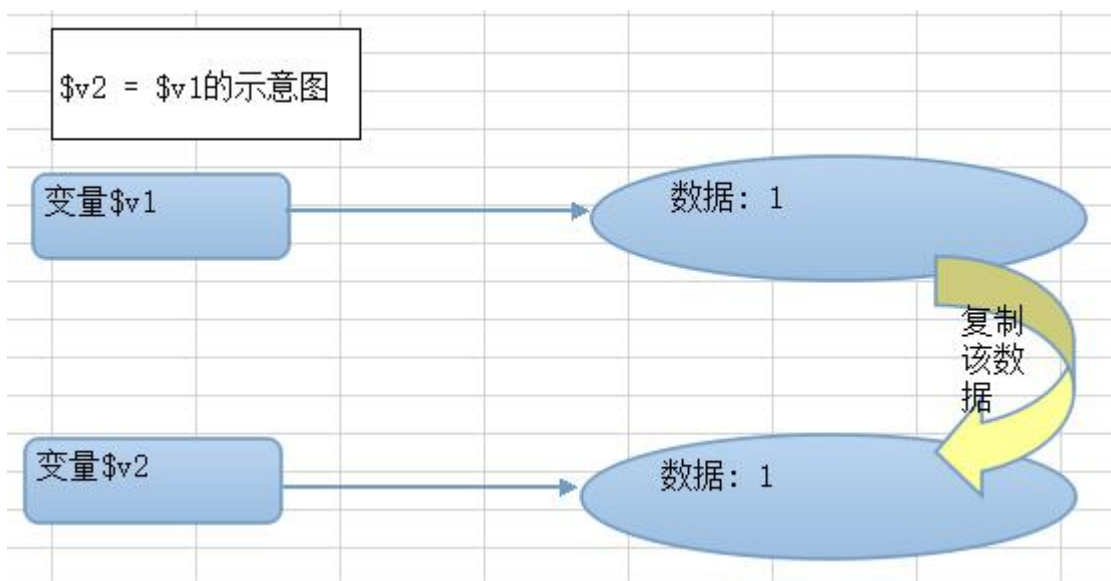
`$v1 = 1;`

```
$v2 = $v1;
```

可见：

值传递之后，两个变量互不影响，相互独立，没有关系了！

其原理如下所示：



可见，此时两个变量在赋值之后，就相互独立，各自有自己的存放空间。改变一个，另一个不受影响。

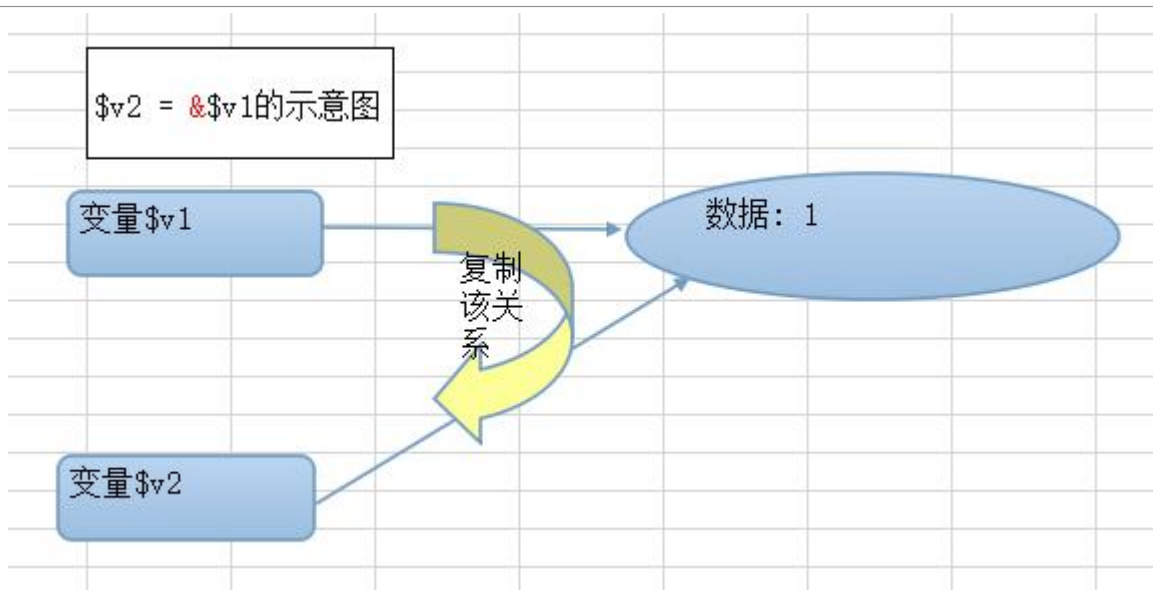
代码演示：

```
01php-yufa.php x 02bianliang-base.php x 03bianliang-isset-unset.php x 04zhichuandi.php x
1 <?php
2 //值传递：
3 $v1 = 10;
4 $v2 = $v1; //这就叫做“传值”
5 // $v3 = $v1 + 1; //这不叫传值！
6
7 echo "<br>v2为：", $v2; //10
8 $v1 = 11; //改变v1的值！
9 echo "<br>v2为：", $v2; //10
10
11 $v1 = 22; //再次改变v1的值！
12 echo "<br>v2为：", $v2; //10
13 /*
14 值传递总结：
15 1. 在传值的那个时刻，两个变量的值一样（相等）
16 2. 在传值完成之后，两个变量没有关系（各自独立）
17 */
18 >>
```

v2为：10
v2为：10
v2为：10

2.4.3. 引用传递：

是将右边变量对数据的引用关系，传给左边的变量。



代码演示:

```
05yinyongchuandi.php x 01php-yufa.php x 02bianliang-base.php x 0
1 <?php
2 //演示引用传值的效果:
3 $v1 = 10;
4 $v2 = &$v1; //赋值时加了一个引用符号"&"
5 //这也是“传值”——引用传值:
6 echo "<br>v2为: ", $v2;
7
8 $v1 = 11; //改变$v1的值
9 echo "<br>v2为: ", $v2;
10
11 $v2 = 22; //改变$v2的值
12 echo "<br>v1为: ", $v1;
13 /*
14 引用传值总结:
15 1, 在引用传值的时刻, 两个变量的值一样
16 2, 在之后, 两个变量的也是一样:
17 改变其中任何一个, 另一个也改变!
18 */
19
```

v2为 : 10
v2为 : 11
v1为 : 22

2.5. 预定义变量

在 PHP 语言内部，有一些（也就 10 来个）变量，是现成的，直接可以使用，这就是所谓预定义变量。我们要做的是事情就是：理解该变量是什么意思，以及怎么用！

2.5.1. \$_GET 变量

代表浏览器表单通过“get”方式提交的所有数据（集），可以称为“get 数据”。

也可以理解为：

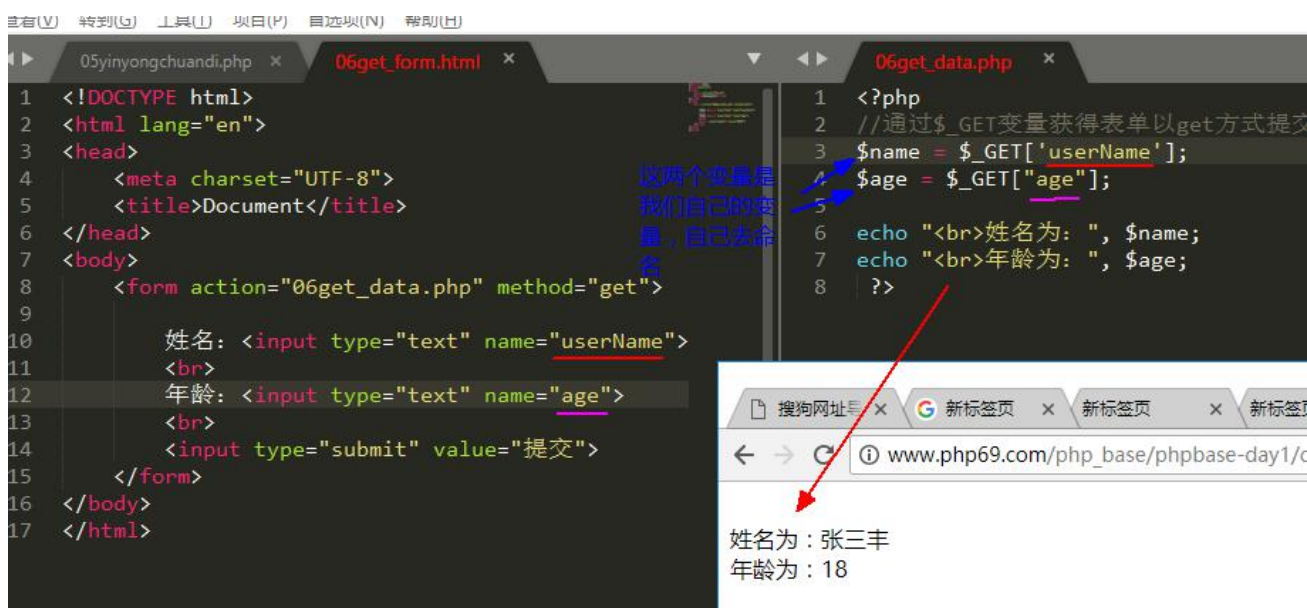
\$_GET 变量里面会“自动存储”（保存/装载）提交到某个文件中的 GET 数据。

而 GET 数据，是在一个页面以“get”方式请求的时候提交的数据。

代码演示：

制作一个表单，有两个输入框，可以输出数字，并提交。

到另一个页面（文件）中，计算这两个数字的和！



对该加法计算的一个改进：

2.5.2. \$_POST

代表浏览器表单通过“post”方式提交的所有数据（集），可以称为“POST 数据”。

也可以理解为：

\$_POST 变量里面会“自动存储”（保存/装载）提交到某个文件中的 POST 数据。

而 POST 数据，是在一个表单中以“post”方式提交的数据。

代码演示：

有个表单，两个输入框，可以填写数字，有一个“提交按钮”，点击提交，可以计算他们的和并输出。



```

查看(V) 转到(G) 工具(T) 项目(P) 首选项(N) 帮助(H)
05yinyongchuandi.php x 07post_form.html x 07post_data.php x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8     <form action="07post_data.php" method="post">
9
10         数字1<input type="text" name="num1">
11         <br>
12         数字2<input type="text" name="num2">
13         <br>
14         <input type="submit" value="计算!">
15     </form>
16
17 </body>
18 </html>
1
2 <?php
3 //通过$_GET变量获得表单以get方式提交的数据!
4 $n1 = $_POST['num1'];
5 $n2 = $_POST['num2'];
6 $result = $n1 + $n2;
7 echo "相加计算的结果为: ", $result;
8 ?>

```

扩展（提交给自己！）：

```

05yinyongchuandi.php x 08post_form.php x
7
8 <?php
9 $n1 = ""; // 初始化这3个变量，以供后续去直接输出
10 $n2 = "";
11 $result = "";
12 if( isset($_POST['num1']) ) // 判断该变量中的数据是否存在
13 {
14     $n1 = $_POST['num1'];
15     $n2 = $_POST['num2'];
16     $result = $n1 + $n2;
17     //echo "<br>计算结果为: " , $result;
18 }
19 ?>
20
21 <body>
22 <!-- action留空表示提交给自己（本页）-->
23 <form action="" method="post">
24
25     数字1<input type="text" name="num1" value="<?php echo $n1; ?>"> // 输出
26     <!-- 有兴趣有时间的同学，可以把这个"+"用一个select来代替，用户可以选择加减乘除 -->
27     +
28     数字2<input type="text" name="num2" value="<?php echo $n2; ?>"> // 输出
29     <input type="submit" value="">
30     <input type="text" name="result" value="<?php echo $result; ?>"> // 输出
31 </form>
32 </body>
33 </html>

```

2.5.3. \$_REQUEST

代表浏览器通过“get”方式 或 “post”方式提交的数据的合集。

即：它既能接收到 get 过来的数据，也能接收到 post 过来的数据！

通常，一个表单，只提交一种形式的数据，要么 get 数据，要么 post 数据！

代码演示：

但，有一个情况，提交 post 数据的同时，也可以提交 get 数据：



```

7
8 <?php
9 $n1 = "";
10 $n2 = "";
11 $result = "";
12 if( isset($_REQUEST['num1']) )
13 {
14     $n1 = $_REQUEST['num1'];
15     $n2 = $_REQUEST['num2'];
16     $result = $n1 + $n2;
17     //echo "<br>计算结果为: " , $result;
18
19     echo "<br>用户名: " , $_REQUEST['userName'];
20     echo "<br>你的账户余额为: 0,请交费。。。";
21 }
22 ?>
23
24 <body>
25     <form action="09request_form.php?id=10&userName=zhangsan" method="post">
26         <!-- 这种表单形式，才可以让一个表单同时提交get数据和post数据:
27             action中的地址里的? 后的数据是get数据
28             form表单中的各个表单项，就是post数据! -->
29         数字1<input type="text" name="num1" value="<?php echo $n1; ?>">
30         +
31         <input type="text" name="num2" value="<?php echo $n2; ?>">
32         <input type="submit" value="=">
33         <input type="text" name="result" value="<?php echo $result; ?>">
34     </form>
35 </body>

```

这属于get数据

这属于post数据

课堂练习:

一个页面中有一个表单：可以输入年龄，并进行判断：
如果大于 18 岁，可以上网吧，
否则不可以！



查找(I) 查看(V) 转到(G) 工具(T) 项目(P) 首选项(N) 帮助(H)

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Document</title>
6  </head>
7  <?php
8
9  //如果有age数据提交过来
10 if(isset( $_POST['age'] ) )
11 {
12     $age = $_POST['age'];//取得该age数据
13     //然后判断其大小
14     if($age > 18)
15     {
16         echo "可以上网";
17     }
18     else
19     {
20         echo "no! ";
21     }
22 }
23 ?>
24 <body>
25     <form action="" method="post">
26         你贵庚: <input type="text" name="age">
27         <input type="submit" value="提交">
28     </form>
29 </body>
30 </html>

```

可以上网
你贵庚:

2.5.4. \$_SERVER 变量

它代表任何一次请求中，客户端或服务端的一些“基本信息”或系统信息，包括很多（10 多项）。我们无非就是要知道，哪些信息是可以供我们使用的！

常用的有：

PHP_SELF:	表示当前请求的网页地址（不含域名部分）
SERVER_NAME:	表示当前请求的服务器名
SERVER_ADDR:	表示当前请求的服务器 IP 地址
DOCUMENT_ROOT:	表示当前请求的网站物理路径（apache 设置站点时那个）
REMOTE_ADDR:	表示当前请求的客户端的 IP 地址
SCRIPT_NAME:	表示当前网页地址

代码演示：



```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8 <?php
9 // PHP_SELF:          表示当前请求的网页地址（不含域名部分）
10 // SERVER_NAME:       表示当前请求的服务器名
11 // SERVER_ADDR:       表示当前请求的服务器IP地址
12 // DOCUMENT_ROOT:     表示当前请求的网站物理路径（apache设置站点时那个）
13 // REMOTE_ADDR:       表示当前请求的客户端的IP地址
14 // SCRIPT_NAME:       表示当前网页地址
15 echo "<br>你访问的网页为: " , $_SERVER['PHP_SELF'];
16 echo "<br>你访问的网站为: " , $_SERVER['SERVER_NAME'];
17 echo "<br>你访问的网站物理路径为: " , $_SERVER['DOCUMENT_ROOT'];
18 echo "<br>你的IP为: " , $_SERVER['REMOTE_ADDR'];
19 $ip = $_SERVER['REMOTE_ADDR'];
20
21 file_put_contents("$ip.txt", $ip)

```

必须学会查手册：

The image shows the PHP manual's Table of Contents. On the left, the 'Language Reference' (语言参考) section is highlighted with a red circle. Below it, the 'Predefined Variables' (预定义变量) section is also highlighted. On the right, the 'Table of Contents' lists various superglobal variables, with '\$_SERVER' circled in red. The list includes:

- [超全局变量](#) — 超全局变量是在全部
- [\\$GLOBALS](#) — 引用全局作用域中可
- [\\$_SERVER](#) — 服务器和执行环境信
- [\\$_GET](#) — HTTP GET 变量
- [\\$_POST](#) — HTTP POST 变量
- [\\$_FILES](#) — HTTP 文件上传变量
- [\\$_REQUEST](#) — HTTP Request 变
- [\\$_SESSION](#) — Session 变量
- [\\$_ENV](#) — 环境变量
- [\\$_COOKIE](#) — HTTP Cookies



2.6. 可变变量

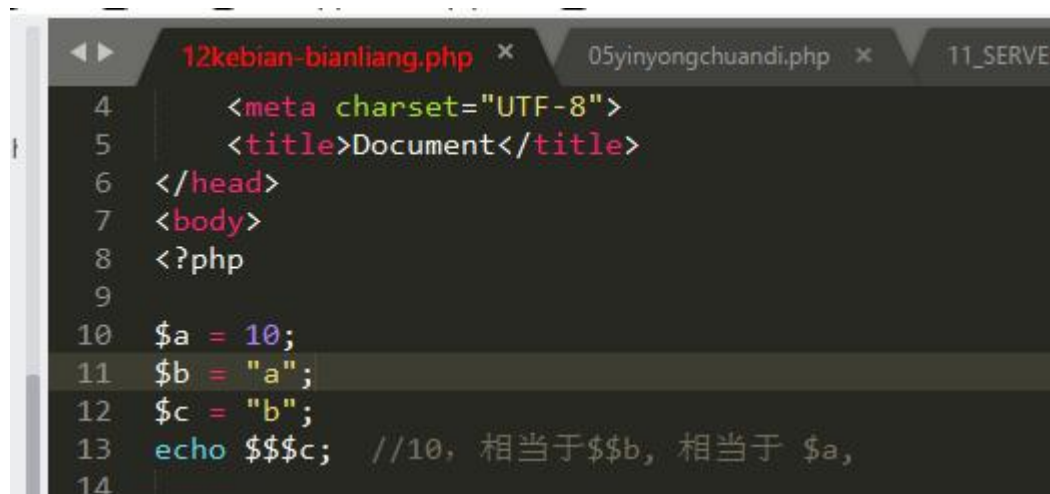
含义：

变量名本身又是一个“变量”的变量。

示例：

```
$v1 = 10;
echo $v1; //输出 10
$str = "v1"; //这是一个变量，名为 str，值为“v1”（字符串）
echo $$str; //输出 10。这里，“$$str”就是所谓的“可变变量”
//相当于 echo $v1; 结果就是 10
```

代码演示：



```
12kebian-bianliang.php x 05yinyongchuandi.php x 11_SERVER
4 <meta charset="UTF-8">
5 <title>Document</title>
6 </head>
7 <body>
8 <?php
9
10 $a = 10;
11 $b = "a";
12 $c = "b";
13 echo $$c; //10, 相当于$$b, 相当于 $a,
14
```

3. 常量

3.1. 常量的含义

常量，就是一个用于存储“不会（也不允许）变化的数据”的标识符。

比如圆周率，在一定的应用场景中，就是一个固定的值（人为规定为某个值）。



3.2. 常量的两种定义形式

3.2.1. define()函数形式:

define(‘常量名’, 对应的常量值);

常量名推荐使用“全大写”。

3.2.2. const 关键字定义:

const 常量名 = 对应的常量值;

演示:

3.3. 常量的两种取值形式

3.3.1. 直接使用:

echo 常量名; //注意：使用常量，前面不带”\$”符号，也不能有引号

3.3.2. 使用 constant()函数以取值:

echo constant(‘常量名’); //注意，此时常量名要用引号引起来。



```

2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8 <?php
9 // 定义常量的形式1
10 define("PI1", 3.14);
11 // 定义常量的形式2
12 const PI2 = 3.1416;
13
14 // 计算半径为2的圆的面积，用PI1作为圆周率
15 $s1 = PI1 * 2 * 2; // 使用形式1
16 // 计算半径为3的圆的面积，用PI2作为圆周率
17 $s2 = PI2 * 3 * 3;
18 // 计算半径为4的圆的面积，用PI1作为圆周率
19 $s3 = constant("PI1") * 4 * 4; // 使用形式2
20
21 echo "<br>面积1为: ", $s1;
22 echo "<br>面积2为: ", $s2;
23 echo "<br>面积3为: ", $s3;
24
25

```

3.4. 变量与常量的区别：

- 1，变量的数据可以变化（重新赋值），常量不可以。
- 2，变量可以存储各种数据类型，而常量只能存储简单数据类型。
- 3，给变量赋值可以是“计算”结果，但给常量赋值，只能是“直接写出的值”（字面值）

```

$vl = 1+2; //对的
const P1 = 1+2; //错的！

```

3.5. 判断一个常量是否存在：defined();

判断的结果返回：true（表示存在）或 false（表示不存在）

形式：

```

if ( defined('常量名') ) {    //如果该常量名存在，则....
    //....做什么事情。。。
}

```

演示：



```
29 // 如果常量PI1没有定义过，
30 if( !defined("PI1") )
31 {
32     //那么这里就去定义它！
33     define("PI1", 3.14);
34 }
35 // 计算半径为2的圆的面积，用PI1作为圆周率
36 $s1 = PI1 * 2 * 2; //这里就直接使用该常量
37
```

3.6. 预定义常量

预定义常量就是 PHP 语言内部预先定义好的常量，我们可以直接使用。

比如：PHP_VERSION, PHP_OS, PHP_INT_MAX, M_PI 等。

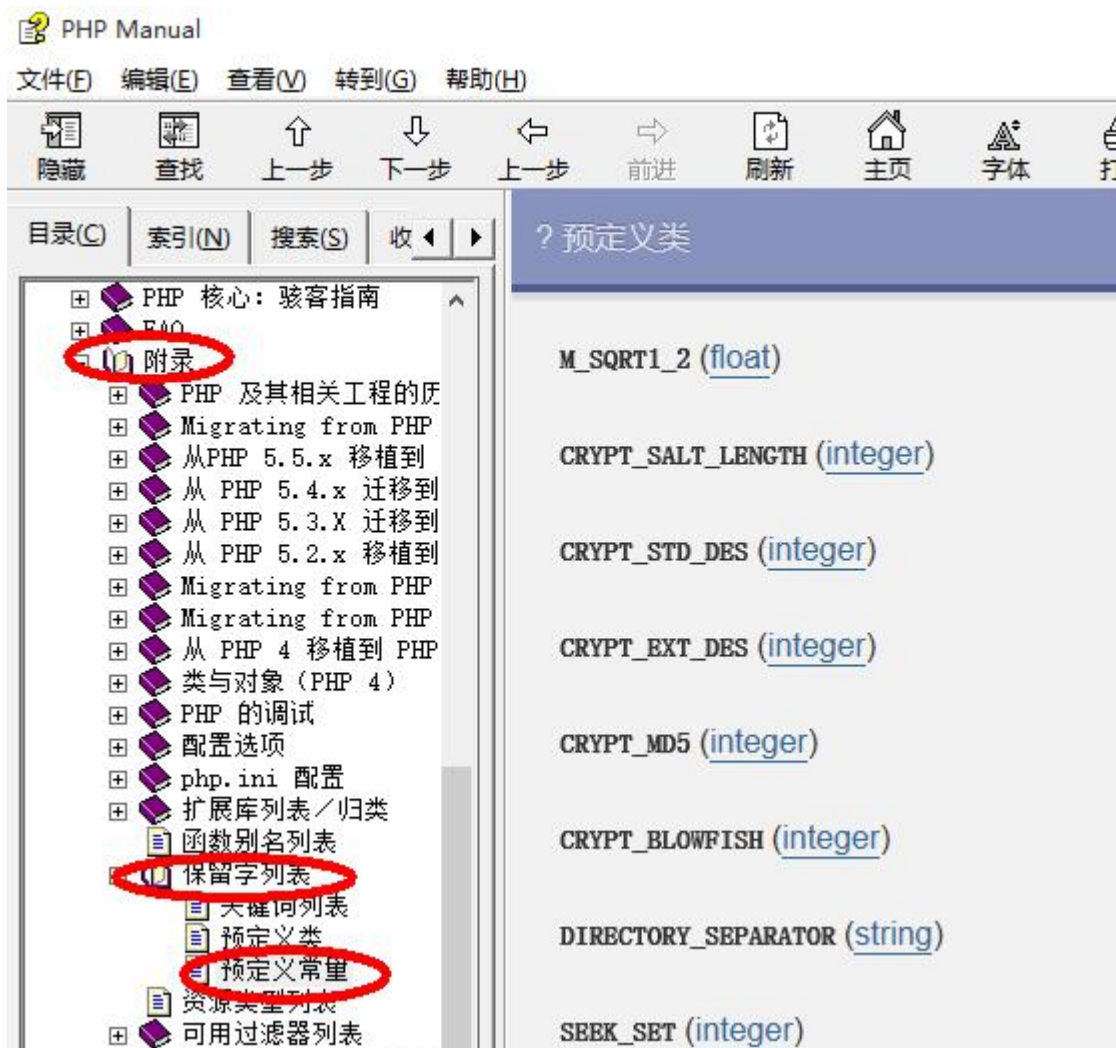
PHP_VERSION: 表示当前 php 的版本信息

PHP_OS: 表示当前 php 运行所在的系统信息

PHP_INT_MAX: 表示当前版本的 php 中的最大的整数值

M_PI: 表示圆周率 π （一个有 10 多位小数的数）

查手册，预定义常量如下：



3.7. 几个魔术常量

含义：

魔术常量也是常量，只是在形式上为常量，而其值其实是“变化”的。

他们也是系统中预先定义好的，也就几个，下面是最常用的 3 个：

__DIR__ ：代表当前 php 网页文件所在的目录
__FILE__ ：代表当前 php 网页文件本身的路径
__LINE__， ：代表当前这个常量所在的行号

```
<?php
//演示3个预定义常量:
echo "<br />当前网页所在目录: " . __DIR__ ; 当前网页所在目录: H:\itcast\class\bj-quanzhan4\day2
echo "<br />当前网页本身的路径" . __FILE__ ; 当前网页本身的路径H:\itcast\class\bj-quanzhan4\day2\11pre_constant.php
echo "<br />当前这一行的行号: " . __LINE__ ; 当前这一行的行号: 12
echo "<br />当前这一行的行号: " . __LINE__ ; 当前这一行的行号: 13
echo "<br />当前这一行的行号: " . __LINE__ ; 当前这一行的行号: 14
```



今日总结

变量：

用于存储数据的一个“标识符”——就是变量名。

变量的命名规则：

基本规则：

以字符，下划线，数字构成，并不能以数字开头

行业规则：

尽量见名知意。

大驼峰命名法，小驼峰命名法

变量的4种基本操作：

赋值，取值，判断 `isset()`，销毁 `unset()`

变量的传值方式：

概念：将一个变量传给另一个变量。

值传递：各自独立，互不干扰。

引用传递（传地址，地址传递）：他们都指向同一个数据！

改变其中任何一个，另一个也跟着改变。

但：`unset()`其中任何一个，另一个不受影响。

4个预定义变量：

`$_GET`, `$_POST`, `$_REQUEST`,

`$_SERVER`：一些在请求中的客户端或服务端的固定的数据。

比如：`REMOTE_ADDR`, `SERVER_ADDR`, `DOCUMENT_ROOT`, `PHP_SELF`

可变变量：

`$$v1`;

常量：

定义形式：

`define(“常量名”, 值);`

`const 常量名 = 值;`

`const C1 = 1+2;`

使用形式：

直接使用常量名：

`echo PI;`

`constant(“常量名”)`

`echo constant('PI');`

判断常量是否存在：`defined(“常量名”)`

预定义常量：

`PHP_INT_MAX`（php中的整数的最大值）

魔术常量：

`__DIR__`

`__FILE__`

`__LINE__`



day1 结束

4. 数据类型

4.1. 数据类型分类与概述

4.1.1. 标量类型：

标量类型也可以理解为“基本类型”，“简单类型”。

标量类型包括如下 4 种：

字符串类型： string

就是一串字符，当做一个整体，表示一个连续有确定顺序的字符串。

整数类型： integer, int

浮点数： double, float

就是数学上的小数。

布尔类型： boolean, bool

表示某种只有两个状态（可能值）的数据，比如性别，是否毕业，吃没吃饭；

4.1.2. 复合类型：

就是“数组”（array）和“对象”（object）两种。

4.1.3. 特殊类型：

空类型 null： 有且只有一个数据可以使用，那就是 null

资源类型 resource： 表示一种外部的可供 php 使用（操作）的资源（比如图片，数据库，文件等）

4.2. 整型 integer/int

4.2.1. 四种书写形式

可以有4种直接书写的形式：

10 进制形式：

```
$n1 = 123;
```

8 进制形式：

```
$n1 = 0123; //以0开头，只能出现0-7这8个数字
```

16 进制形式：

```
$n1 = 0x12A34; //以0x开头，可以出现0-9，A-F这16个数字
```

2 进制形式：

```
$n1 = 0b1011011010; //以0b开头，只能出现0和1这2个数字
```

下表为进制对比：

10 进制	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8 进制	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20
16 进制	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2 进制	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000

演示：

```
08post_form.php x 01int-base.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6 </head>
7 <body>
8 <?php
9   $n1 = 123;
10  $n2 = 0123;
11  $n3 = 0x123;
12  $n4 = 0b1010;
13  echo "<br>n1=" . $n1;
14  echo "<br>n2=" . $n2;
15  echo "<br>n3=" . $n3;
16  echo "<br>n4=" . $n4;
```

Document
n1=123
n2=83
n3=291
n4=10



4.2.2. 进制的相互转换

直接通过系统函数来进行，能完成：

10 进制转为 2, 8, 16 进制：

decbin(): 将 10 进制转为 2 进制

decoct(): 将 10 进制转为 8 进制

dechex(): 将 10 进制转为 16 进制

或：

2, 8, 16 进制转为 10 进制：

bindec(): 将 2 进制数字字符串转为 10 进制

octdec(): 将 8 进制数字字符串转为 10 进制

hexdec(): 将 16 进制数字字符串转为 10 进制

演示：

```
08post_form.php x 01int-base.php
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6 </head>
7 <body>
8 <?php
9 $n1 = 123;
10 $n2 = 0123;
11 $n3 = 0x123;
12 $n4 = 0b1010;
13 echo "<br>n1=" . $n1;
14 echo "<br>n2=" . $n2;
15 echo "<br>n3=" . $n3;
16 echo "<br>n4=" . $n4;
```

n1=123
n2=83
n3=291
n4=10

总结一下进制转换的单词：

dec: 10 进制

oct: 8 进制

hex: 16 进制

bin: 2 进制

进制转换案例：

表单上有个输入框，一个下拉列表（包含 6 种转换），以及一个按钮“转换”。实现可灵活进行进制转换的功能。



```

7  <?php
8  $n1 = "";
9  $n2 = "";
10 $zh = "";
11 if(isset($_POST['submit1'])){
12     $n1 = $_POST['n1'];
13     $zh = $_POST['zhuanhuan'];
14     if( $zh == "10to2" )
15     {
16         $n2 = decbin($n1);
17     }
18     if( $zh == "10to8" )
19     {
20         $n2 = decoct($n1);
21     }
22 }
23
24 ?>
25 <body>
26     <form action="" method="post">
27         <input type="text" name="n1" value="<?php echo $n1;?>">
28         <select name="zhuanhuan">
29             <option value="10to2" <?php if($zh == "10to2"){ echo "selected"; } ?> >10to2</option>
30             <!-- <option value="10to8" 如果$zh等于"10to8"就输出selected >10to8</option> -->
31             <option value="10to8" <?php if($zh == "10to8"){ echo "selected"; } ?> >10to8</option>
32         </select>
33         <input type="submit" name="submit1" value="转换">
34         <input type="text" name="n2" value="<?php echo $n2;?>">
35     </form>
36 </body>

```

思考题：

怎么将一个 8 进制的数， 转换为 16 进制？

4.3. 浮点型 double/float

浮点数就是相当于数学上的“小数”

两种书写形式：

常规形式（带小数点）：

\$f1 = 0.1; //或者 1.23; 123.0;

科学计数法形式（带 e）：

\$f2 = 1.23e3; //表示 1.23 乘以 10 的 3 次方

\$f3 = 123e2; //这个也是浮点数，虽然其结果值是一个“整数”（12300）

特别注意：浮点数不要随便做相等比较（==）：因为浮点数进行相等比较，是“不可靠”的：

\$v1 = 0.1 + 0.2;

\$v2 = 0.3;

问：\$v1 和 \$v2 相等吗？

那怎么办？

需要在考虑精度的基础上，将浮点数转换为整数，然后进行“比较大小”，此时，在精度范围内，如果相等，我们就认为是相等的。将上述程序改造为：



演示：

```
01int-base.php x 05float.php x
4 <meta charset="UTF-8">
5 <title>Document</title>
6 </head>
7 <body>
8 <?php
9 $v1 = 0.1;
10 $v2 = 0.2;
11 $v3 = 0.3;
12 if( $v3 == $v1 + $v2)
13 {
14     echo "相等1";
15 }
16 else
17 {
18     echo "不相等1";
19 }
20 echo "<h1>下面来对其做“正确的比较方式”</h1>";
21 //假设目前需要精确到小数点后4位,
22 //那么就乘以10000, 然后转换为整数去比较
23 if( round($v3 * 10000) == round( ($v1 + $v2) * 10000 ) )
24 {
25     echo "相等2";
26 }
27 else
28 {
29     echo "不相等2";
30 }
31
```

Document x

← → ↻ ① www.php69.com/php_base/phpbase-day2/code/05fl

不相等1

下面来对其做“正确的比较方式”

相等2

4.4. 布尔型 boolean

只有两个数据值： true， false（不区分大小写）

在需要当做“布尔值”进行判断或比较的时候，以下数据会被当作布尔值的 false：

false, 0, 0.0, “”, null, ‘0’, 空数组

其余都被当做布尔值的 true。

常见应用代码类似这样：

```
$v1 = 某某值;
if( $v1 ){ //此时，如果$v1 为上述值之一，就会当做 false，不执行其中的代码。
    ... 代码...
}
//而对于其他值，就会当做 true，因此就会执行其中的代码。
```




4.5. 字符串型 string

可以使用单引号或双引号来表示（引起来）。

表示一串“连续的字符”，最短的字符串是“空字符串”，比如：\$str1 = ""。

注意：这个不是空字符串：“ ”， 因为其中包括了 1 个空格字符。

\$name = “张三”;

\$edu = ‘北京大学’;

\$title = “关于微信应用的几个改进建议”; //文章标题

\$v1 = ‘;’; //这也是字符串（空字符串），跟 null 不同!!!

\$v2 = ‘123’; //这还是字符串

\$v3 = “false”; //这仍然是字符串

特别注意：

双引号字符串中，如果出现“\$”符号，则该符号后的连续字符（单词）会被识别为一个变量名。

如果识别成功（即确实存在该变量），则会将该变量转换为实际内容。

如果识别失败（即实际没有该变量），则会报错。

演示：

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8 <?php
9 $v1 = 10;
10 echo "<br>v1的值为: " . $v1;
11 echo "<br>v1的值为: $v1";
12 echo "<br>v1的值为: $v1";
13 echo "<br>";
14 echo "<br>\$v1的值为: $v1";
15

```

Document

www.php69

v1的值为: 10
v1的值为: 10
v1的值为: \$v1
\$v1的值为: 10

4.6. 数组类型 array

数组，就是将多个“数据”放在一起，排成一个有序序列，这个序列作为一个整体（里面包括了多个数据），就称为“数组”。比如：

\$info1 = array(‘张三丰’, 18, “男”);

//新版本中，还可以这样写： [‘张三丰’, 18, “男”]

或：

\$info2 = array(‘name’=> ‘张三丰’, ‘age’=>18, ‘gender’=> “男”);



//新版本中，还可以这样写： ['name'=>'张三丰', 'age'=>18, 'gender'=>"男"]

数组的赋值：

\$数组变量[下标] = 一个值；

数组的取值：

形式为：\$数组变量[下标]

比如：echo \$info1[0]; print_r(\$info2['name']);

数组的每一项，都是由“下标”和对应的“值”构成，其内部结构，如下所示：

\$info1 内部结构：

键名/下标/索引	0	1	2
值	'张三丰'	18	'男'

这种下标，通常称为“数字下标”。

\$info2 内部结构：

键名/下标/索引	'name'	'age'	'gender'
值	'张三丰'	18	'男'

这种下标，通常被称为“字符串下标”。

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Document</title>
6 </head>
7 <body>
8 <?php
9 //定义数组
10 $info2 = array('name' => '张三丰', 'age'=>18, 'gender' => "男" );
11
12 //给数组某项赋值
13 $info2['age'] = 19;      //重新对该数组的age这一项赋值
14
15 //使用数组（使用单项）
16 echo "姓名: " . $info2['name'];
17 echo "<br>年龄: " . $info2['age'];
18 //使用数组（使用整体）:
19 echo "<br><pre>";
20 print_r($info2);      //打印（输出）该数组
21 echo "</pre>";
22
23 ?>
24 </body>

```

Document

www.php69.c

姓名：张三丰
年龄：19

Array
(
 [name] => 张三丰
 [age] => 19
 [gender] => 男
)

4.7. 空类型 NULL

只有一个值： null（不区分大小写）。



`$v1 = null;` //此时，具有变量`$v1`，但其中的数据是“空的”（没有数据）
空值变量，`isset()`判断的结果是 `false`（即不存在）。

4.8. 类型判断

`gettype()`: 获取一个变量的类型，结果为一个变量类型的名称（字符串）

```
$v1 = 10;
$r1 = gettype( $v1 );    //结果为: "integer"
```

```
$v2 = 'abc';
$r2 = gettype( $v2 );    //结果为: "string"
```

```
$v3 = 1.23;
$r3 = gettype( $v3 );    //结果为: "double" (返回的是 double，但其实就是我们所说的 float 类型)
```

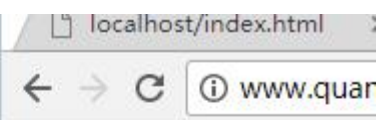
`settype()`: 设置一个变量的类型

```
$v1 = 10;    //此时，$v1 中数据是整数类型
settype( $v1, 'string' ); //此时，$v1 是字符串类型，即其中的数据变成了: "10"
echo gettype( $v1 );    //"string"
```

PHP 是一门“弱类型语言”！

```
$v1 = 10;    //此时 v1 是 int 类型
$v1 = 'abc'; //此时 v1 就是 string 类型
```

`var_dump()`: 输出变量的“完整信息”，包括变量类型，长度（如果需要），内容



```
$v1 = 1;
$v2 = 2.2;
$v3 = 'abcde';
$v4 = true;
var_dump($v1);
echo "<br />";
var_dump($v2);
echo "<br />";
var_dump($v3);
echo "<br />";
var_dump($v4);
```

int(1)
float(2.2)
string(5) "abcde"
bool(true)

判断是否为某种类型（类型系列函数）：

`is_int()` / `is_integer()`: 判断是否为整数类型



`is_float()`: 判断是否为浮点类型

`is_bool()`: 判断是否为布尔类型

`is_string()`: . . .

`is_array()`: . . .

`is_numeric()`: 判断是否为“数字”类型（含整数，小数，以及“纯数字字符串”）

`is_object()`:

两个特殊判断:

`isset()`: 判断一个变量是否存在，或变量中是否有数据，有则返回 `true`，否则返回 `false`。

如果变量中有数据，返回 `true`

如果变量中没有有数据，返回 `false`

`empty()`: 判断是否为“空的”。如果确实是“空的”，返回 `true`，否则返回 `false`。

如果变量中的数据为空的，返回 `true`

如果变量中的数据不是空的，返回 `false`

“空的”的意思，比较接近日常生活中的“没有”。以下值都是空的（`empty`）:

0, 0.0, “”, “0”, `false`, `array()`, //这几项，`empty` 判断的结果都是 `true`

而空（`null`）是一个计算机中的特殊概念，表示“完全不存在”，可以理解为“真空”。

4.9. 类型转换

4.9.1. 自动转换

我们无需做任何处理，而是，程序会根据运算时运算符所需要的数据类型进行转换。

如果参与运算的数据不是需要的类型，则会自动转换为需要的数据类型。

`$v1 = 1 + "3";` //结果是 4;

`$v2 = 1 . "3";` // “.” 是字符串连接符，这里，1 会被转换为字符串，结果是：“13”

典型自动转换:

转换为数字:

`1+"2"` //3

`"1" + "2"` //3

`1 + "2abc"` //3

`1 + "2abc34"` //3

`1 + "abc"` //1

`1 + "abc2"` //1

`1.2 + "2"` //3.2

`1.2 + "2.2abc"` //3.4

`1.2 + "abc2.2"` //1.2

`"1.2abc" + 2` //3.2

`"1.2abc" + "2abc"` //3.2

规律：一个字符串当做数字，就会将该字符串的最前面的数字转换为数字值，如果没有，就为 0 转换为整数（直接保留整数部分）:

`10.8 % 3.6`

`"10.8" % "3.6"`



“10.8” % “3.6abc”

“10.8ab” % “3.6cd”

4.9.2. 强制转换:

人为使用强转换语法进行转换，比如：

\$v1 = (int) “1”; //结果，\$v1 是整数类型的 1

\$v2 = (float) “1.23”; //结果，\$v2 是浮点类型的 1.23

\$v3 = (string) \$v1; //结果，\$v3 是字符串 “1”

5. 运算符详解

5.1. 概述:

5.1.1. 含义

就是对数据进行某种所需要的运算的语法符号，比如加减乘除，或比较大小，或判断真假。

5.1.2. 按参与运算的数据的个数来分类

单目运算符：

只需要一个数据——但必须是变量。

双目运算符：

需要两个数据——可以是变量，也可以直接的数据本身。

三元运算符：

需要 3 个数据才能运算，也称为三目运算符。

只有一个三元运算符。

5.1.3. 按功能分类

赋值运算符：

=

算术运算符：

+ - * / %

连接运算符：

.

自赋值运算符：

+= -= *= /= %= .=



自操作运算符：

++ --

比较运算符：

> >= < <= == != === !==

逻辑运算符：

&&(与) || (或) ! (非)

条件运算符：

数据 1 ? 数据 2 : 数据 3

位运算符：

& | ~

其他：

@, 是错误抑制符

() , 括号, 用于提升运算优先级, 括号中的先运算。

5.2. 赋值运算符

只要理解一个核心的观念：

就是将等号(=)右边的数据(可能是运算结果数据)赋值给左边的变量。

看看以下写法：

\$v1 = 1;

\$v2 = 2;

\$v3 = \$v1 + \$v2;

\$v1 + \$v2 = \$v3; //可以吗? 不可以!

if(\$v1 + \$v2 = \$v3){...} //可以吗? ? ? 还是不可以!

一定记住, 一个等号的左边, 只能是一个“变量名”!

所以, 这个写法是错误的: \$v1+2 = \$v2 + 3

5.3. 算术运算符

针对数字进行的算术计算, 包括: + - * / %

%: 对整数数字进行“取余操作”——如果不是整数, 会先自动转换为整数之后再进行取余。

转换为整数的做法是: 直接去掉小数部分

比如:

\$v1 = 10 % 4; //结果为 2

\$v2 = 10.8 % 3.6 //结果为 1

如果不是数字, 会自动转换为数字进行。

举例:

\$v1 = 3 + “4”; //7

\$v2 = “3” + “4”; //7

\$v3 = “3” + “4.5”; //7.5



```
$v4 = 3 + "4abcd"; //7
$v5 = "3ab" + "4cd"; //7
$v6 = 3 + "abc4"; //3
$v7 = "ab3" + "cd4"; //0
$v8 = "ab" + "cd"; //0
$v9 = 3 + true; //4 true 转为数字，为 1
$v10 = 3 + false; //3 false 转为数字，为 0
```

演示：

```

6 </head>
7 <body>
8 <?php
9
10 $v2 = 10.8 % 3.9; //结果为1
11 var_dump($v2);
12 ?>
13 </body>

```

5.4. 连接运算符(.)

就是字符串的连接，能够将前后字符连接起来。

如果不是字符串，会自动转换为字符串。

```
$v1 = "ab" . "cd"; // 'abcd'
$v2 = $v1 . "ef"; // 'abcdef'
$v3 = "ab" . 12; // "ab12"
$v4 = 12 . 34; // "1234"
$v5 = "12" . true; // "121"，true 转为字符串，为 "1"；
$v6 = "12" . false; // "12"，false 转为字符串，为 ""（即空字符串）；
```

5.5. 自赋值运算符

针对数字的，包括： += -= *= /= %=

针对字符串的，只有一个： .=

形式：

\$变量 += 数据;

相当于这个赋值语句：**\$变量 = \$变量 + 数据;**

```
$v1 = 1;
```

```
$v1 += 2; //3, 相当于: $v1 = $v1 + 2;
```



此时，其实是相当于该变量的值，跟给定的数据进行运算后的结果数据，再放回到该变量中（覆盖了之前的数据）。

演示：

```

9
10 $v1 = 'abc';
11 $v1 .= 'def'; 这里进行的是连接运算
12 echo "<br>v1为: $v1";
13 $v1 += 2; 这里进行的是加法
14 echo "<br>此时v1为: $v1";
15

```

v1为 : abcdef
此时v1为 : 2

5.6. 自操作(自加自减) 运算符

针对数字，只有 2 个：++ --

属于单目运算符，即只要一个变量就可以进行运算。

形式：

\$变量++; //对该变量中数据+1
 \$变量--; //对该变量中数据-1
 ++\$变量; //对该变量中数据+1
 --\$变量; //对该变量中数据-1

++讨论：

表示对该变量进行“自加 1”操作。即该变量中的数据加 1。

前自加：先自加，后取值；

后自加：先取值，后自加；

举例 1：

```

8 <?php
9
10 $n1 = 10;
11 $n2 = 10;
12 $n1++; //自加
13 ++$n2; //自加
14 echo "<br>n1加加后: " . $n1;
15 echo "<br>n2加加后: " . $n2;
16 //结论: "++"运算，都是对相应的变量加1
17 //不管是放在前面(前++), 还是后面(后++)
18

```

n1加加后 : 11
n2加加后 : 11

常见自加与赋值的混合运算：

\$v1 = 1;



```
$v2 = $v1++;  
$s1 = 1;  
$s2 = ++$s1;
```

```
24 $m1 = 10;  
25 $m2 = 10;  
26 $s1 = $m1++; //这是后自加：先取值，再自加，再去进行其他运算  
27 $s2 = ++$m2; //这是前自加：先自加，后取值，再去进行其他运算  
28 echo "<br>此时，m1为：$m1"，"，s1为：$s1";  
29 echo "<br>此时，m2为：$m2"，"，s2为：$s2";  
30 echo "<br>m1:"，$m1++ + 5;  
31 echo "<br>m2:"，++$m2 + 5;  
32  
33 echo "<hr>";  
34 echo "<br>m1:$m1";  
35 echo "<br>m2:$m2";
```

此时，m1为：11，s1为：10
此时，m2为：11，s2为：11
m1:16
m2:17
m1:12
m2:12

总结：

- 1，前自加，后自加，对变量本身的结果都是加1，没有区别。
- 2，但前自加、后自加如果跟别的运算符一起运行，此时区别为：
 - 前自加：先自加，后取值（然后去进行其他运算）；
 - 后自加：先取值，后自加（然后去进行其他运算）；

5.7. 比较运算符

含义：

是针对数字的大小进行比较的运算。

如果不是数字，会自动转换为数字。

包括：> >= < <= == != === !==

所有比较运算符，运算的结果只有两个可能（之一）：true 或 false

比如：

```
if( 1 == 2); //结果是 false
```

等于（==）和全等于（===）的区别：

通常，等于（==）也会用“模糊相等”或“松散相等”的说法。

两个数据“基本相等”（类型可能会发生自动转换），就算是相等。

```
1 == "1" //true  
0 == false; //true  
true == 1 //true  
2.0 == 2; //true
```

全等于（===）就是完全相等：

只有两个数据的数据类型一样，并且其值也一样的时候，才是全等。

```
1 === "1" //false
```



```

0 === false;    //false
true === 1      //false
2.0 === 2;      //false
1 !== "1"       //true
0 !== false;    //true
true !== 1      //true
2.0 !== 2;      //true
    
```

查手册》附录》类型比较表：

松散比较 ==												
	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

严格比较 ===												
	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
"php"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

不等于： !=

两个数据不满足“==”这个运算结果，不等于（!=）的结果就是 true，比如：



```
if ( 1 != 2 )    //true
```

不全等于：！==

两个数据不满足“==”这个运算结果，不全等于（！==）的结果就是 true，比如：

```
if ( “1” != 1 ) //true
```

5.8. 逻辑运算符

针对“布尔值”进行的运算，只有 3 个：

与： &&

或： ||

非： !

如果不是布尔值，会自动转换为布尔值去进行运算。

因为布尔值只有 2 个，不管怎么排列组合，最终只有以下 10 种情况的计算，几乎就是公式化。

5.8.1. 逻辑与（&&）：

含义：

表示两个条件（数据）的真假结果是否同时为真的共同作用结果。

运算结果：只有两个条件都为真的时候，结果才是真（true）。

只有如下 4 种情况：

true && true	结果为 true
true && false	结果为 false
false && true	结果为 false
false && false	结果为 false

总结归纳出逻辑与（&&）的真值表（就是公式，类似 99 乘法表）：

\$v1, \$v2 表示任意变量（条件）	\$v1 = true 时	\$v1 = false 时
\$v2 = true 时	\$v1 && \$v2 : true	\$v1 && \$v2 : false
\$v2 = false 时	\$v1 && \$v2 : false	\$v1 && \$v2 : false

可见：逻辑与运算的结果，只有两个是真，才是真。

实际应用中，往往是以“条件”的面目出现，而不是简单的真假值。

案例：

给定一个任意整数，请判断该数是否能被 3 整除并且还能被 5 整除！

```
$v1 = 18;
```

```
if( $v1 % 3 == 0 && $v1 % 5 == 0 ){....}
```




```

7  <?php
8  /*
9   给定一个任意整数，请判断该数是否能被3整除并且还能被5整除！
10 */
11 $n1 = 15;    //比如13,19,20,21,
12 if( $n1 % 3 == 0 && $n1 % 5 == 0)
13 {
14     echo "ok, 满足条件";
15 }
16 else
17 {
18     echo "no, 不满足条件";
19 }
20
21 ?>
22 <body>

```

5.8.2. 逻辑或（||）：

含义：

表示两个条件（数据）的真假结果的是否存在“真”的情况的共同作用结果。

运算规则：

只要有任意一个条件为真（true），结果就是真。

只有如下4种情况：

true true	结果为 true
true false	结果为 true
false true	结果为 true
false false	结果为 false

总结归纳出逻辑或（||）的真值表（就是公式，类似99乘法表）：

\$v1, \$v2 表示任意变量（条件）	\$v1 = true 时	\$v1 = false 时
\$v2 = true 时	\$v1 \$v2 : true	\$v1 \$v2 : true
\$v2 = false 时	\$v1 \$v2 : true	\$v1 \$v2 : false

可见：逻辑或运算的结果，只要有一个是真，就是真。

实际应用中，往往也是以“条件”的面目出现，而不是简单的真假值，比如：

\$v1 = 18; //请判断该数是否能被3整除或能被5整除！

if(\$v1 % 3 == 0 || \$v1 % 5 == 0){....}

案例：

给定一个整数表示年份，请判断该年份是否为闰年。

（注：闰年是能被4整除但不能被100整除的年，或者能被400整除的年。）

<body>

<form action="" method="post">

请输入年份：<input type="text" name="year">



```
<input type="submit" value="判断">
</form>

<?php
    if( !empty($_POST['year']))
    {
        $year = $_POST['year']; //永远是字符串！
        if(is_numeric($year))
        {
            $year2 = (int)$year;
            if($year2 == $year)
            {
                if($year % 4 == 0 && $year % 100 != 0 || $year % 400 == 0)
                {
                    echo $year . "是闰年。";
                }
                else
                {
                    echo "{$year}不是闰年。";
                }
            }
            else
            {
                echo "年份应该为整数!";
            }
        }
        else
        {
            echo "请输入整数!";
        }
    }
    else
    {
        echo "<font color='red'>请输入年份</font>";
    }
?>

</body>
</html>
```

5.8.3. 逻辑非（！）：

就是对一个布尔值进行“取反”操作，规则为：

!true 结果为 false

!false 结果为 true

逻辑非（！）在 if 中的常见应用情形：

if(!isset(\$v1)){...} //如果\$v1 为空（null）

if(!empty(\$v1)){...} //如果\$v1 不为空（empty）

对应的两个相反的情形是：

if(isset(\$v1)){...} //如果\$v1 存在

if(empty(\$v1)){...} //如果\$v1 为空（empty）

区分两个“空”：

null：“真空”，什么都没有，是明确定义的“没有数据”的一种写法（含义）

empty：“现实意义上的没有”，就是无，就是 0,等等，常见的，认为是 empty 的数据有：

“”（空字符串），0, 0.0, “0”, false, null, 空数组，

还是查手册》附录》类型比较表：

使用 PHP 函数对变量 \$x 进行比较					
表达式	gettype()	empty()	is_null()	isset()	boolean : if(\$x)
\$x = "";	string	TRUE	FALSE	TRUE	FALSE
\$x = null;	NULL	TRUE	TRUE	FALSE	FALSE
var \$x;	NULL	TRUE	TRUE	FALSE	FALSE
\$x is undefined	NULL	TRUE	TRUE	FALSE	FALSE
\$x = array();	array	TRUE	FALSE	TRUE	FALSE
\$x = false;	boolean	TRUE	FALSE	TRUE	FALSE
\$x = true;	boolean	FALSE	FALSE	TRUE	TRUE
\$x = 1;	integer	FALSE	FALSE	TRUE	TRUE
\$x = 42;	integer	FALSE	FALSE	TRUE	TRUE
\$x = 0;	integer	TRUE	FALSE	TRUE	FALSE
\$x = -1;	integer	FALSE	FALSE	TRUE	TRUE
\$x = "1";	string	FALSE	FALSE	TRUE	TRUE
\$x = "0";	string	TRUE	FALSE	TRUE	FALSE
\$x = "-1";	string	FALSE	FALSE	TRUE	TRUE
\$x = "php";	string	FALSE	FALSE	TRUE	TRUE
\$x = "true";	string	FALSE	FALSE	TRUE	TRUE

5.8.4. 逻辑运算的短路规则：

短路的基本概念：

就是对于“逻辑与”或“逻辑或”这两种运算符，他们可能会只进行左侧的逻辑判断之后，立即就



中断后续（右侧）的判断，而得出整个逻辑运算符的运算结果！

逻辑与的短路：

if(左侧判断 && 右侧判断) //此时，如果左侧判断为 false，则右侧判断不再进行

逻辑或的短路：

if(左侧判断 || 右侧判断) //此时，如果左侧判断为 true，则右侧判断不再进行

示例：

假设给定一个整数表示一个员工的年龄，如果该年龄为整十的数，或者年龄的平方除以 100 在 10~20 之间，则公司在当年年底会给该员工发特别奖。请写出程序来判断某个年龄的员工是否可以得奖。

```
8  <?php
9  // 假设给定一个整数表示一个员工的年龄，
10 // 如果该年龄为整十的数，
11 // 或者年龄的平方除以100在10~20之间，
12 // 则公司在当年年底会给该员工发特别奖。
13 // 请写出程序来判断某个年龄的员工是否可以得奖。
14 $age = 20; //当age为20,30,就会发生短路
15 if($age % 10 == 0 || $age*$age/100>=10 && $age*$age/100<= 20)
16 {
17     echo "<br>可以得奖";
18 }
19 else
20 {
21     echo "<br>不可以得奖";
22 }
23
24 echo "<br>再来一个写法： ";
25 //这个，对于32也会短路
26 if($age*$age/100>=10 && $age*$age/100<= 20 || $age % 10 == 0)
27 {
28     echo "<br>可以得奖";
29 }
30 else
31 {
32     echo "<br>不可以得奖";
33 }
34
```

结论：

我们应该将我们可能发生短路运算的逻辑判断的前后判断语句，做如下调整：

相对简单的运算放前面（左侧），相对复杂的运算放后面（右侧）。

对于逻辑或的短路运算，道理一样。

昨日回顾



类型判断：

gettype(): 获得一个变量的类型名称，结果就是类型名称的字符串。
 settype(\$变量, 目标类型): 设置一个变量的类型：相当于改变了变量的类型。
 var_dump();
 is_int(), is_float(), is_string(), is_bool(), is_array(), is_object(), is_null(),
 is_numeric():

类型转换：

自动转换：由运算符来决定
 强制转换：（目标类型）数据

运算符：

赋值。。。
 算术运算符：
 连接运算符：
 自赋值运算符：
 += -= *= /= %= .=
 \$v1+= 10; // \$v1 = \$v1+10;
 自操作运算符：
 ++:

比较运算符：

== : 相等，松散相等，模糊相等
 === : 全等于，类型一致，数据相等

逻辑运算符：

&&: 只要有一个是 false，结果就是 false
 || : 只要有一个是 true，结果就是 true
 ! :
 短路现象：

5.9. 条件运算符

只有一个条件运算符，形式为：

表达式 1 ? 表达式 2 : 表达式 3

含义：

如果表达式 1 为 true（或自动转换后为 true），则运算的结果值为表达式 2，否则为表达式 3。

举例：

```
$score = 77; //分数
$v1 = $score >= 60 ? “及格” : “不及格”; //结果为“及格”
```

其本质是：

```
$score = 77; //分数
if ( $score >= 60 ){
    $v1 = “及格” ;
}
```




```
else{
    $v1 = “不及格” ;
}
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
</head>
<body>
<?php

$year = 2003; //年份
$is_runnian = ($year % 4 == 0 && $year % 100 != 0 || $year % 400 == 0) ? "是闰年" : "不是闰年";
echo $year,$is_runnian;

?>
</body>
</html>
```

Document

www.php69.com/php_base/phpbase-day3/code/04tiaojian-yunsuanfu.php

2003不是闰年

5.10. 位运算符（了解）

针对整数进行的二进制级别的运算。

基本位运算符包括：按位与（&），按位或（|），按位非（~），按位异或（^）

5.10.1. 位运算基本运算规则：

位运算符有如下基本运算规则：

（只针对二进制的 0 和 1 这两个数据的基本位运算规则）：

按位与（&）运算规则：

0 & 1 结果为：0

0 & 0 结果为：0

1 & 0 结果为：0

1 & 1 结果为：1

结论：只有两个都是 1，按位与运算结果才是 1

按位或（|）运算规则：

0 | 1 结果为：1

0 | 0 结果为：0

1 | 0 结果为：1

1 | 1 结果为：1

结论：只有两个都是 0，按位或运算结果才是 0

按位非（~）：

~1 结果为 0

~0 结果为 1

按位异或（^）：

0 ^ 1 结果为：1



$0 \wedge 0$ 结果为：0

$1 \wedge 0$ 结果为：1

$1 \wedge 1$ 结果为：0

规则是：相同为 0，不同为 1

5.10.2. 整数的按位与（&）运算

含义：

是针对整数的二进制值进行的位运算结果。

将两个整数的二进制值的每一个对应位上的二进制数字进行对应的按位与运算

$\$v1 = 6 \& 8;$ //这里，虽然是普通整数，但其内部是按该整数的二进制形式进行位运算

运算规则：

将 6 和 8 的二进制数字的每一个对应位上的数字（0 或 1）进行基本的按位与（&）的运算所得到的结果。

数字 6 的二进制	0	0	0	0	0	1	1	0
数字 8 的二进制	0	0	0	0	1	0	0	0
按位与结果	0	0	0	0	0	0	0	0

结果：0

再来一个：

$\$v2 = 7 \& 9;$

数字 7 的二进制	0	0	0	0	0	1	1	1
数字 9 的二进制	0	0	0	0	1	0	0	1
按位与结果	0	0	0	0	0	0	0	1

结果：1

还有整数的其他位运算：

按位或，按位左移，按位右移。

5.10.3. 按位左移运算

$\$v1 = 8 \ll 2;$ //8 是要进行左移运算的整数，2 是表示要进行移动的位数。

含义：

将 8 这个数的二进制形式的数字的所有位都往左边移动 2 位，最右边空出部分填 0（补 0），最左边部分就会有 2 位“冒出去”了，也不要处理（不管），这样移动后，所得到的二进制数就是结果。

分析如下：

8 的二进制	0	0	0	0	1	0	0	0
左移两位后：	0	0	1	0	0	0	0	0

结果： $1 * 2^5 = 32$

再来一个：

$\$v2 = 11 \ll 3$



11 的二进制						1	0	1	1
左移 3 位后:	0		1	0	1	1	0	0	0

结果: $1*2^6 + 0 + 1*2^4 + 1*2^3 + 0 + 0 = 64 + 0 + 16 + 8 = 88$

总结:

将一个整数左移 n 位，就是相当于将该数乘以 2 的 n 次方。

5.11. 错误抑制符@

含义:

在一个表达式出现错误的时候，可以将错误“隐藏”（掩盖）起来（不输出）！

通常，该符号，用于在实际运行环境中的一些条件非我们（程序员）所能控制的情形。

如果出现该情形并报错，则我们可以抑制该错误的显示（只是该错误不显示，不是没有错误了）。

典型应用场景:

```
if( @mysqli_connect('localhost', 'root', '123') ){
    .....
}
```

加上抑制符后:

```
<body>
<?php
//错误抑制符
if(@mysqli_connect('localhost','root','1234'))
{
    echo "成功! ";
}
else{
    echo "连接失败";
}
```

5.12. 运算符的优先级

运算符优先级不得不说的规则:

1, 时刻意识到，运算符有优先级问题！



- 2, 小括号可以改变运算的顺序（即括号最优先）
 - 3, 先乘除（以及取余）后加减；
 - 4, 大致有如此优先级规则：算术运算 > 比较运算 > 逻辑运算
 - 5, 赋值运算符通常都是最后（优先级最低）
- 细致的可以看手册。

6. 流程控制

流程控制，就是研究程序的走向。

6.1. 流程控制概述

6.1.1. 三大流程结构：

顺序结构：

程序运行的自然状态，就是从前往后（从上到下）运行程序。

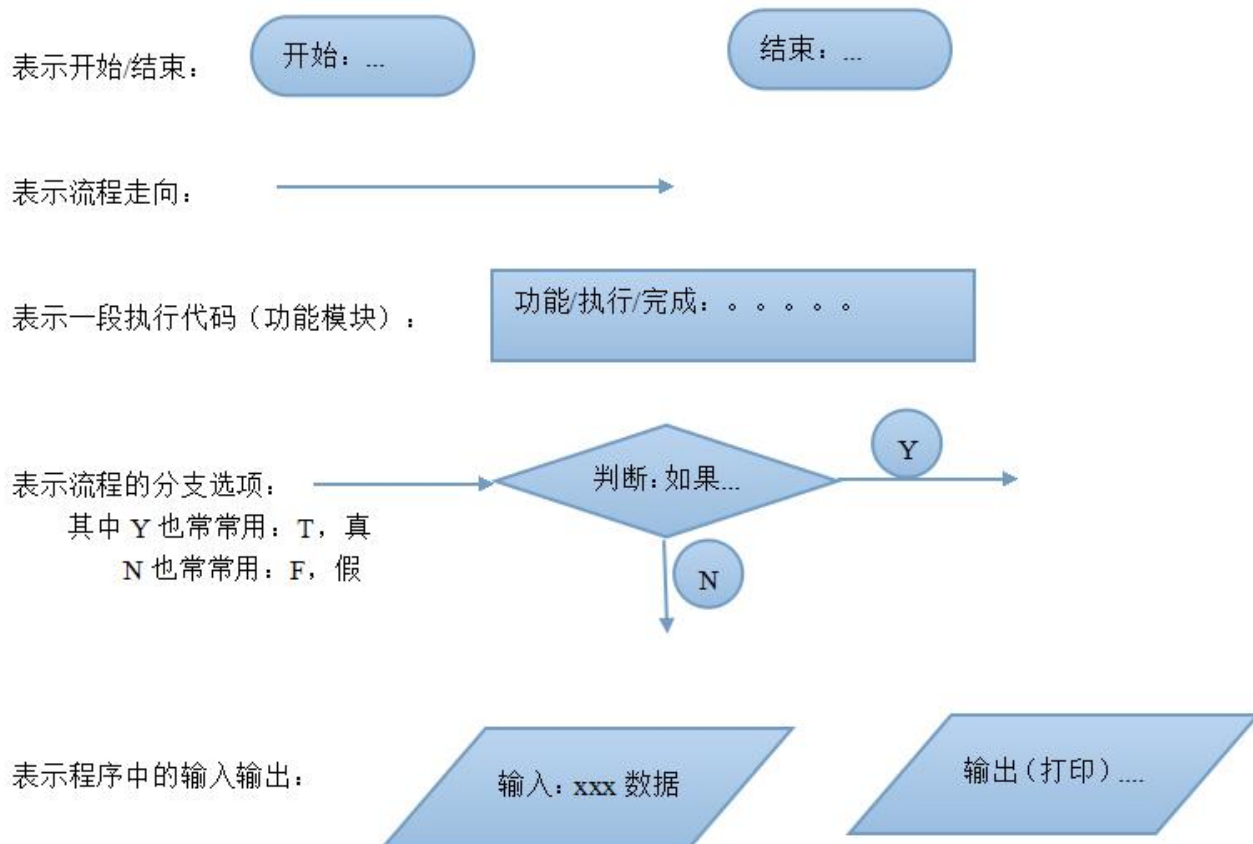
分支结构：

程序运行过程中，根据判断条件的不同结果(true 或 false)，执行不同的分支（其他分支不再执行）。

循环结构：

程序运行过程中，根据判断条件的不同结果(true 或 false)，决定是再次执行还是不再执行。

6.1.2. 流程图常用图形符号：



6.2. if 语句

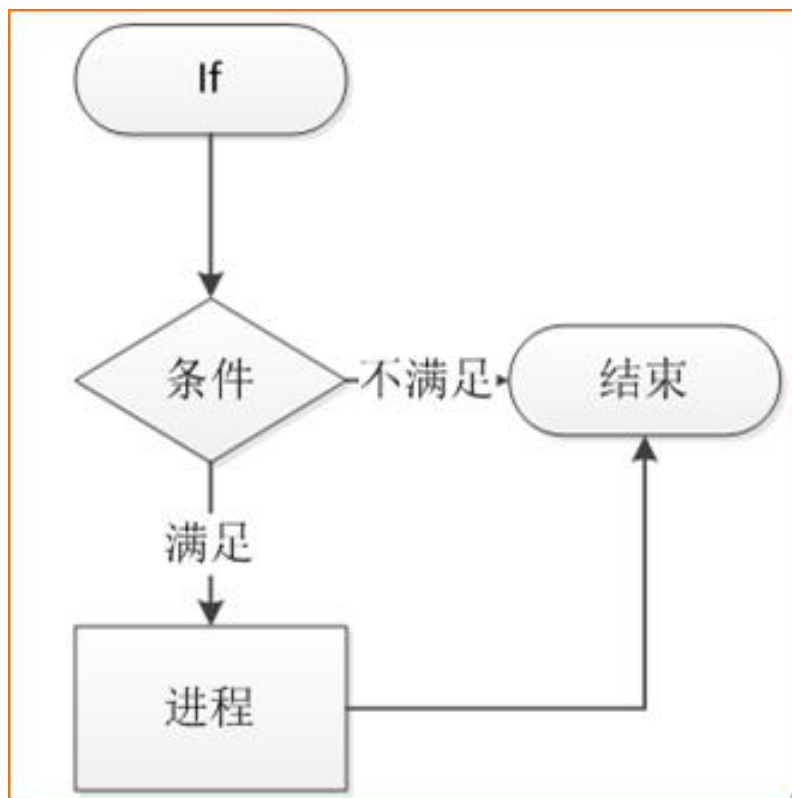
有如下几种常见的 if 语句（不同的分支数量）：

6.2.1. 形式 1：单分支

形式：

```
if( 条件判断 )  
{  
    //如果条件满足，就执行这里  
}
```

流程图：

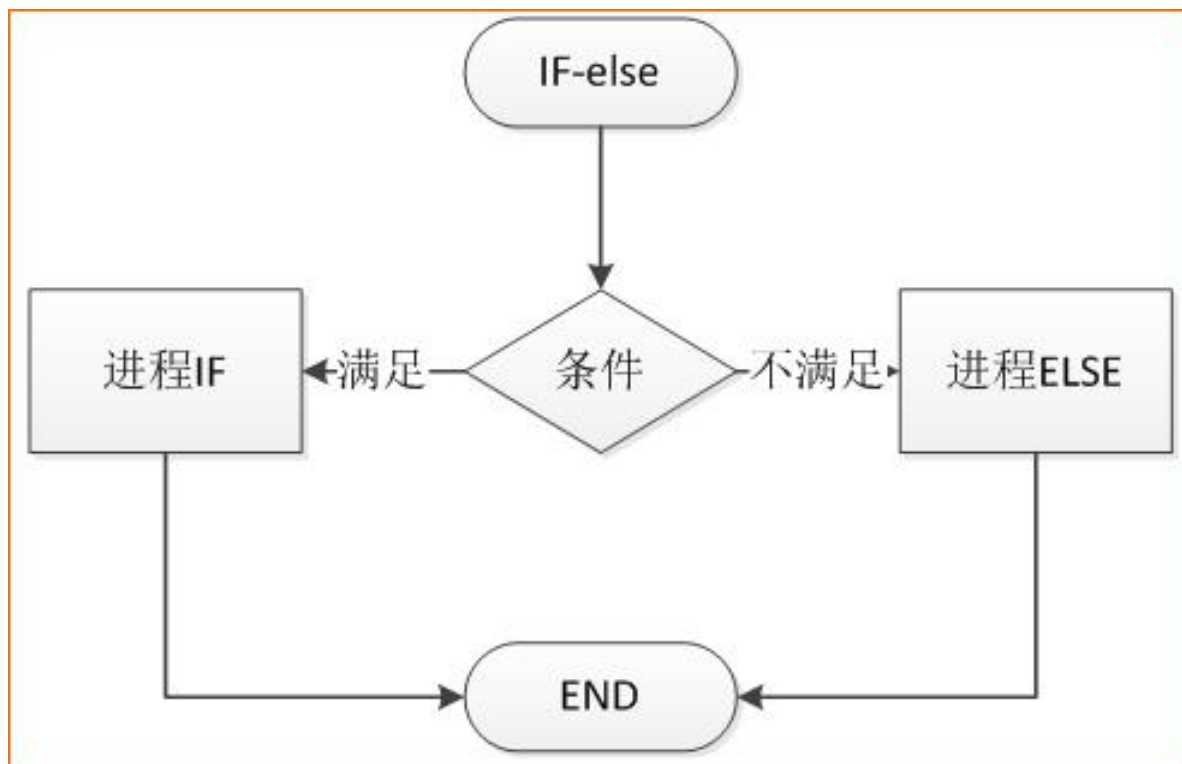


6.2.2. 形式 2：双分支

形式：

```
if( 条件判断 )
{
    //如果条件满足，就执行这里
}
else
{
    //如果条件不满足，就执行这里
}
```

流程图：

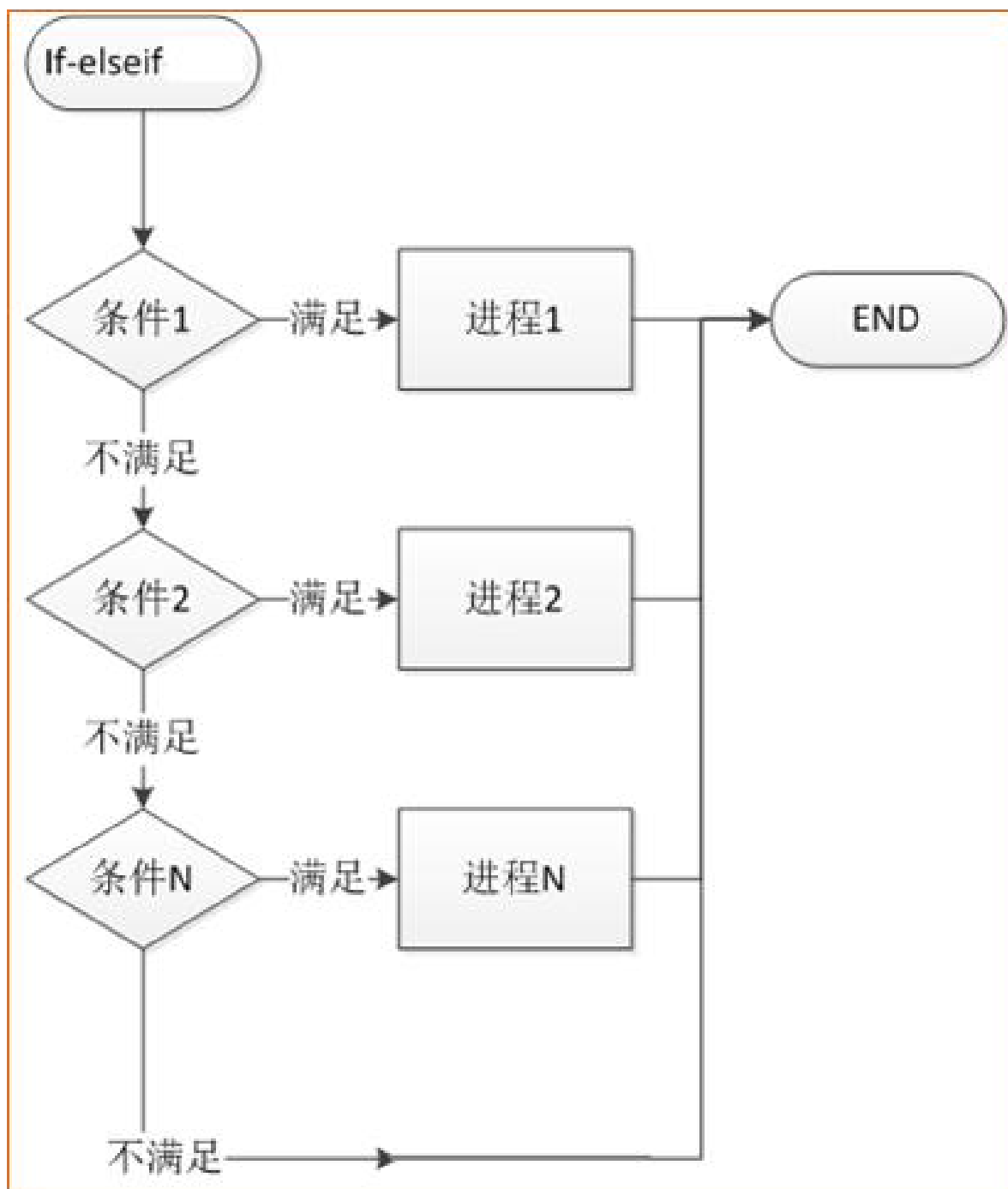


6.2.3. 形式 3：多分枝

形式：

```
if( 条件判断 1 ) {  
    //分支 1;  
}  
elseif( 条件判断 2 ) {  
    //分支 2;  
}  
elseif( 条件判断 3 ) {  
    //分支 3;  
}  
。。。。。。前面的 elseif 分支，可以若干个（0 个以上）
```

流程图：



示例：

如果我有钱，那么我的规划是：

- 有 10 万以上，买一个汽车
- 有 2 万以上，买一个摩托车
- 有 5000 以上，买一个电动单车
- 有 1000 以上，买自行车



```
01anw  02anwei-zuoyi.php  05if-elseif.php x
15  /*
16
17  $money = 999;
18  if( $money >= 100000)
19  {
20      echo "买汽车";
21  }
22  elseif($money >= 20000)
23  {
24      echo "买摩托";
25  }
26  elseif($money >= 5000)
27  {
28      echo "买电动单车";
29  }
30  elseif($money >= 1000)
31  {
32      echo "自行车";
33  }
34  else
35  {
36      echo "搬砖去吧";
37  }
38
```

搬砖去吧

6.2.4. if 语句综合形式

语法:

```
if ( 条件 1 )
{
    分支 1
}
elseif ( 条件 2 )
{
    分支 2;
}
elseif ( 条件 3 )
{
    分支 3;
}
. . . . .
else
{

```



else 分支： //前面没有一个条件满足的时候，就执行本分支

说明：

- 1, 上述，黄色部分，是 if 语句“必不可少”的部分。
- 2, 上述，灰色部分是 elseif 分支，可以重复若干次，也可以没有。
- 2, 上述，茶绿色部分是 else 分支，可以没有。

综合案例：

在一个 form 表单中输入一个分数（0-100 之间），程序可以给该分数进行“评语”，评语包括：优秀，良好，中等，及格，不及格。

```

8      <form action="" method="post">
9          输入分数: <input type="text" name="score"> (0-100之间)
10         <br><input type="submit" value="做评语">
11     </form>
12     <h1>
13     <?php
14         if(!empty( $_POST['score'] ))//如果不为空
15         {
16             $sc = $_POST['score'];
17             if( $sc >= 90 )
18             {
19                 echo "优秀";
20             }
21             elseif($sc >= 80)
22             {
23                 echo "良好";
24             }
25             //... 更多elseif分支，以及else情况
26             //注意:
27             //还要考虑分数不在这个范围的情况
28             //还要考虑不是分数（比如字母）的情况
29         }
30         elseif(isset($_POST['score']) && $_POST['score'] == "0")
31         {
32             echo "不及格";
33         }
34         else{
35             echo "请输入分数";
36         }
37     ?>

```

6.3. 分支结构之 switch 分支语句（重点）

语法：

```
switch ( 一个变量数据或表达式结果$vl )
{
    case 值 1:           //如果$vl 等于 这个“值 1”，就执行本分支
        分支语句 1;
        break;           //表示跳出该分支，也就是跳出 switch 语句。
    case 值 2:           //如果$vl 等于 这个“值 2”，就执行本分支
        分支语句 2;
        break;
    . . . . .           //可以更多的分支
    default:
        默认分支;         //前面都不满足的时候，就执行这里， default 分支也可以不写
}
```

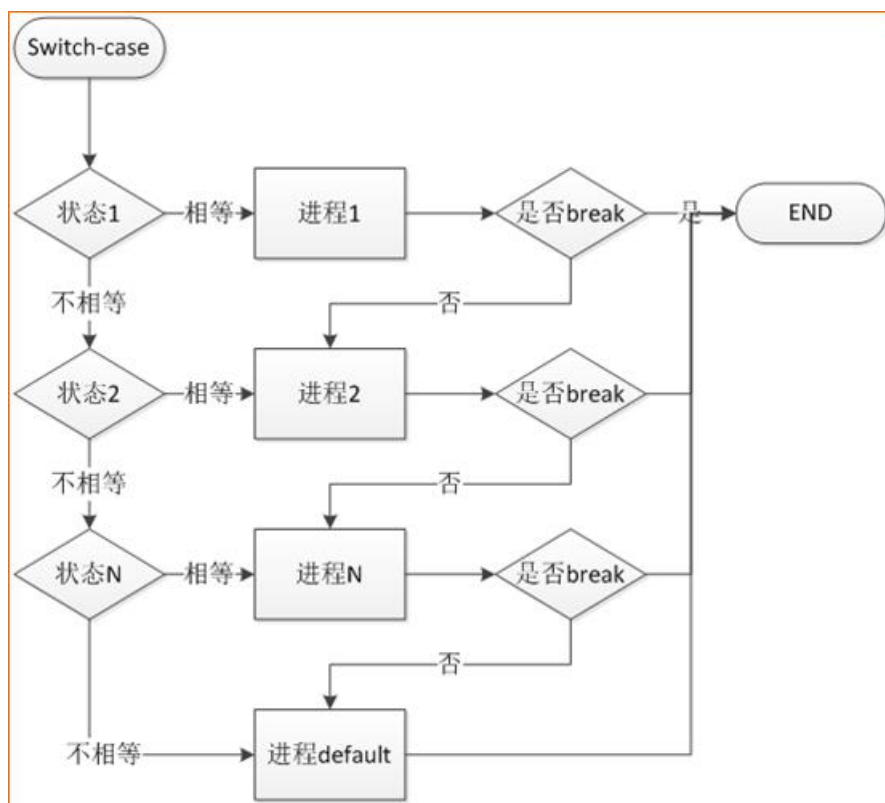
说明：

- 1，上述“值 1，值 2，。。。”可以是各种标量类型，也可以是表达式——因为最终它也是一个值。
- 2，用于进行测试的数据变量\$vl 跟后续的各个值，只能进行“相等比较”（==）。
- 3，从上述第 2 条角度来说，其实际上不如 if 灵活。

注意：

当某个分支满足条件并执行该分支后，如果该分支中没有 break 语句，此时，程序的流程会“直接进入”后一个分支继续执行，直到碰到 break 才会结束 switch。

流程图：



示例：



给出任意年份和月份，输出该月的天数。

代码示例如下：

```
$year = 2008;
$month = 22;
switch( $month )
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        echo "{$year}年{$month}月的天数为 31 天";
        break;
    case 4:
    case 6:
    case 9:
    case 11:
        echo "{$year}年{$month}月的天数为 30 天";
        break;
    case 2:
        //如果是闰年
        if( $year % 4 == 0 && $year % 100 != 0 || $year % 400 == 0 )
        {
            echo "{$year}年{$month}月的天数为 29 天";
        }
        else
        {
            echo "{$year}年{$month}月的天数为 28 天";
        }
        break;
    default:
        echo "没有该月份";
}
```

6.4. 循环结构之 while 循环语句

循环的含义：就是通过某种语法结构，对某段代码可以“反复执行”。

while 循环语法：

```
while (条件判断)
{
    。。。。。循环体语句;
}
```

说明：

循环一开始就进行条件判断：

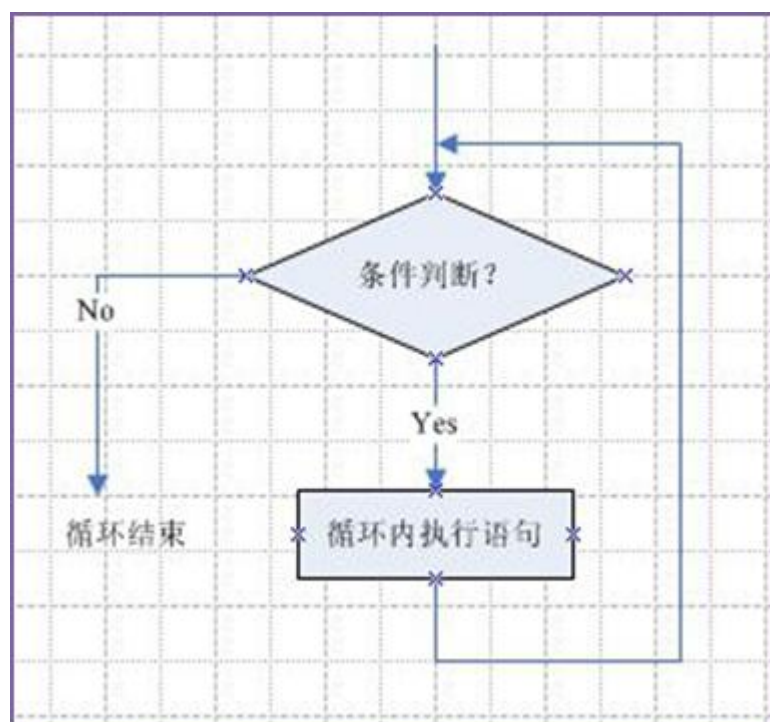
如果成立：则执行循环体，而后会自动回到循环开始位置继续进行条件判断，如此反复；

如果不成立：则退出循环，执行后续语句。

```
10 //循环三要素：
11 //循环变量的初始化
12 $n = 10;
13 while( $n > 5 ) //循环变量的判断（就是条件）
14 {
15     echo "<br>彩票中奖! ";
16     //循环变量的改变
17     $n--;
18 }
19
```

彩票中奖!
彩票中奖!
彩票中奖!
彩票中奖!
彩票中奖!

流程图：



案例：

求 7 到 17 的和。



```

10 //求7到17的和。
11 $n = 7;
12 $sum = 0;
13 while ( $n <= 17)
14 {
15     //$sum = $sum + $n; //好low
16     $sum += $n;
17     $n++;
18 }
19
20 echo "和为$sum";
21

```

和为132

课堂案例：

求 1-100 之间能被 3 整除的数的和！

```

$sum = 0;
$n = 1;
while( $n <= 100)
{
    if( $n % 3 == 0)
    {
        $sum += $n;
    }
    $n++;
}
echo $sum;

```

6.5. 循环结构之 do while 循环语句

do while 循环语法：

```

do
{

    ... 循环体语句;

}while (条件判断);

```

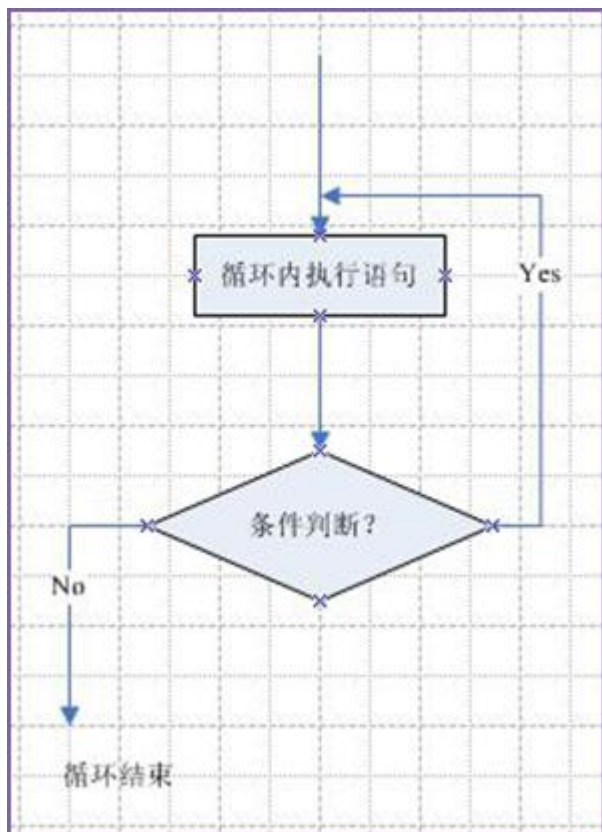
说明：

先执行一次循环体，然后进行条件判断：

如果成立：则继续回去执行循环体，而后再次进行条件判断，如此反复；

如果不成立：则退出循环，执行后续语句。

流程图：



案例：

求 7 到 177 之间能被 7 整除的数的平均值。


```
9 //求7到177之间能被7整除的数的平均值。
10
11 $n = 7;
12 $sum = 0; //总和
13 $count = 0; //个数
14 do
15 {
16     //.....做点什么。。。
17     if($n % 7 == 0)
18     {
19         $sum += $n; //累加器
20         $count++; //计数器
21     }
22     $n++;
23 }while( $n <= 177 );
24
25
26 $avg = $sum / $count; //平均值
27 echo $avg;
28
```

提示:

- 1, 不管是 while, 还是 dowhile, 都请先写出一个良好的可运行的循环结构!
- 2, 通常, 循环体内的最后位置写循环变量的改变语句;

6.6. 循环结构之 for 循环语句（重点/难点）

for 循环语法:

for (循环变量初始化 ①; 循环条件判断 ②; 循环变量的改变 ③) {

..... 循环体语句块 ④;

//这里可以有多条语句

//是可以反复执行的部分

}

说明:

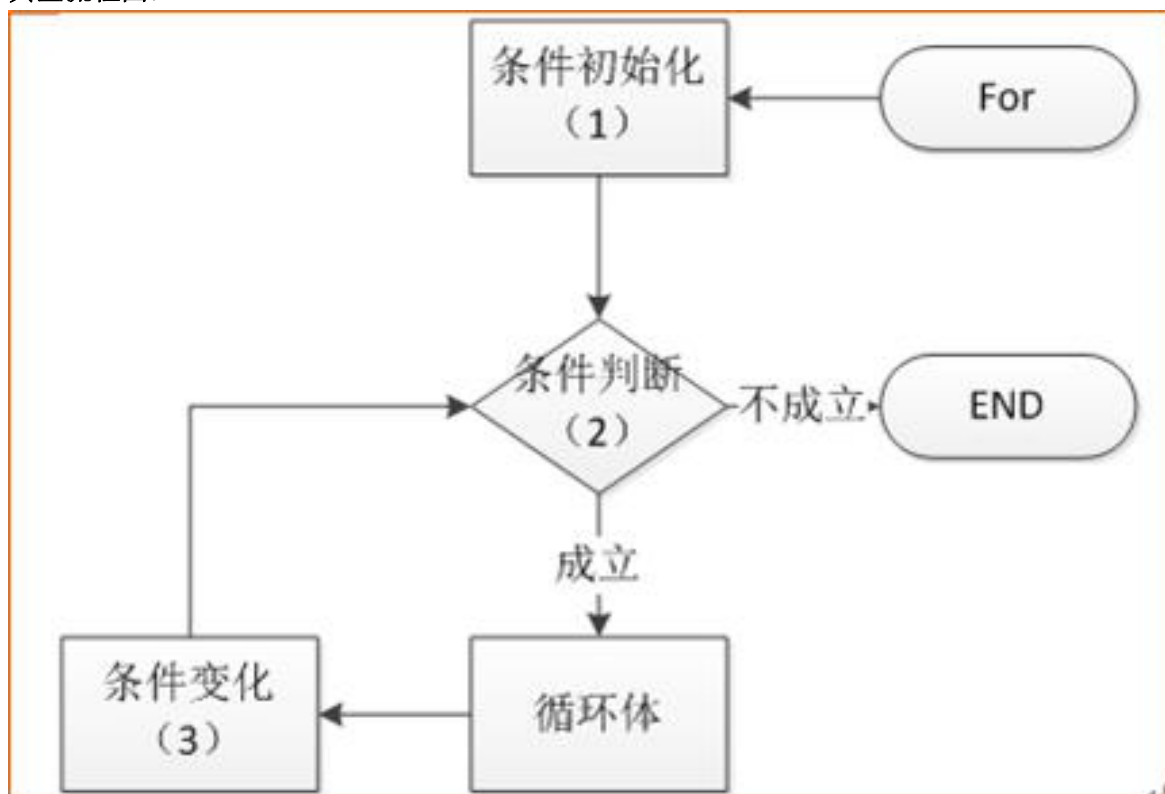
- 1, 执行流程如下图所示



2，此语句结构是将循环的 3 个要素都集中在一起写的形式，比较紧凑，容易控制，最常用。

3，循环变量初始化语句 ① 只执行一次，所以真正的正常循环，是在 “②->④->③” 之间进行。

典型流程图：



案例 1:

求 1-100 之间能被 7 整除的数的和。

```
$sum = 0;    //总和  
for( $i = 1; $i <= 100; $i++)
```



```
{
    if($i % 7 == 0)
    {
        $sum += $i;    //累加
    }
}
echo $sum;
```

案例 2:

输出 1-100 之间能被 3 整除但不能被 5 整除的偶数。

```

9
10 //输出1-100之间能被3整除但不能被5整除的偶数。
11 for($i = 1; $i <= 100; $i++)
12 {
13     if($i % 3 == 0 && $i % 5 != 0 && $i % 2 == 0)
14     {
15         echo "$i, ";
16     }
17 }
18

```

→ 其实是3个条件

Document x TLIAS管理后台

← → ↻ www.php69.com/php_base/phpbase-c

6, 12, 18, 24, 36, 42, 48, 54, 66, 72, 78, 84, 96,

案例 3:

输出 1-100 之间的数，并且要求：

能被 3 整除的使用“三”代替，能被 5 整除的使用“五”代替，能同时被两者整除的用“三五”代替。

```

14 for($i = 1; $i <= 100; ++$i)
15 {
16     if($i % 3 == 0 && $i % 5 == 0)
17     {
18         echo '三五,';
19     }
20     elseif($i % 5 == 0)
21     {
22         echo "五,";
23     }
24     elseif($i % 3 == 0)
25     {
26         echo "三,";
27     }
28     else
29     {
30         echo "$i, ";
31     }
32 }
33

```

Document x TL

← → ↻ www.php69.com/ph

1, 2, 三, 4, 五, 三, 7, 8, 三, 五, 11, 三, 12, 三, 14, 五, 三, 17, 18, 三, 五, 21, 三, 22, 五, 24, 三, 26, 三, 27, 五, 28, 三, 29, 三, 31, 三, 32, 三, 34, 五, 三, 37, 38, 三, 五, 41, 三, 42, 五, 44, 三, 46, 三, 47, 五, 48, 三, 49, 三, 51, 三, 52, 五, 54, 三, 56, 三, 57, 五, 58, 三, 59, 三, 61, 62, 三, 64, 五, 三, 67, 68, 三, 69, 三, 71, 三, 72, 三, 73, 五, 74, 三, 75, 三, 76, 三, 77, 三, 78, 三, 79, 三, 81, 三, 82, 三, 83, 三, 84, 三, 85, 三, 86, 三, 87, 三, 88, 三, 89, 三, 90, 三, 91, 92, 三, 94, 五, 三, 97, 98, 99, 100

课间练习（扩展）：

输出 1-100 之间的带“7”的数，或能被 7 整除的数。



6.7. 多重循环及案例

概念：

多重循环就是循环里面由出现循环的代码。

其基本执行流程，其实仍然是循环的基本逻辑，只是循环的时候要注意：外层循环执行每一次，里层循环就会完整执行循环的“所有次”（一个完整循环的过程）。

案例 1：

输出如下形式的数字到页面上：

```
1 2 3 4 5 6 7 8;
2 2 3 4 5 6 7 8;
3 2 3 4 5 6 7 8;
4 2 3 4 5 6 7 8;
```

```
20 for($n1 = 1; $n1 <= 4; $n1++)
21 {
22     echo "<br>$n1 ";
23     for($n2 = 2; $n2 <= 7; $n2++)
24     {
25         echo "$n2 ";
26     }
27     echo "8; ";
28 }
```

```
1 2 3 4 5 6 7 8;
2 2 3 4 5 6 7 8;
3 2 3 4 5 6 7 8;
4 2 3 4 5 6 7 8;
```

案例 2：

输出如下表格：

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

```
8
9 <?php
10     echo "<table border='1' width=400
11         height=100>";
12     $n = 0;
13     for($i = 1; $i <= 4; $i++)
14     {
15         echo "<tr>"; //输出行的开始标签
16         for($k = 1; $k <= 6; $k++)
17         {
18             $n++;
19             echo "<td>$n</td>";
20         }
21         echo "</tr>"; //输出行的结束标签
22     }
23 ?>
24 </table>
25 <?php
26     echo $n;
27 ?>
```

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

24

案例 3:

实现如下所示 99 乘法口诀表。

九九乘法口诀表

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

```

8  <table border="1" width="700" height="200">
9  <?php
10     for($i = 1; $i <= 9; $i++)
11     {
12         echo "<tr>"; //输出行的开始标签
13         for($k = 1; $k <= $i; $k++)
14         {
15             $s = $k*$i;
16             //echo "<td>{$k}X$i=$s</td>";
17             //上一行还有一种写法如下:
18             echo "<td>{$k}X$i=" . $k*$i . "</td>";
19         }
20         echo "</tr>"; //输出行的结束标签
21     }
22  ?>

```




案例 4:

公鸡 5 元一只，母鸡 3 元一只，小鸡 1 元 3 只，100 元买了 100 只鸡，问各多少只？

```

9 //公鸡5元一只，母鸡3元一只，小鸡1元3只，100元买了100只鸡，问各多少只？
10 /*
11 如果公鸡1只，母鸡1只，小鸡1只，则总价为：..... 不对。
12 如果公鸡1只，母鸡1只，小鸡2只，则总价为：..... 不对。
13 如果公鸡1只，母鸡1只，小鸡3只，则总价为：..... 不对。
14 .....
15 如果公鸡1只，母鸡2只，小鸡1只，则总价为：..... 不对。
16 如果公鸡1只，母鸡2只，小鸡2只，则总价为：..... 不对。
17 如果公鸡1只，母鸡2只，小鸡3只，则总价为：..... 不对。
18 .....
19 如果公鸡100只，母鸡100只，小鸡100只，则总价为：..... 不对。
20 这种编程思想，叫做“穷举”，就是将所有可能的答案都罗列出来，然后挨个去“验证”
21 */
22 $count = 0; //用于记录运行（计算）的次数！
23 for($gongji = 0; $gongji <= 100; $gongji++)
24 {
25     for($muji = 0; $muji <= 100; $muji++)
26     {
27         for($xiaoji = 0; $xiaoji <= 100; $xiaoji++)
28         {
29             if($gongji + $muji + $xiaoji == 100 && $gongji*5 + $muji*3 + $xiaoji/3 == 100)
30             {
31                 echo "<br>公鸡:$gongji, 母鸡:$muji, 小鸡:$xiaoji";
32             }
33             $count++;
34         }
35     }
36 }
37 echo "<br>运行次数: $count";

```

结果:

```

公鸡:0, 母鸡:25, 小鸡:75
公鸡:4, 母鸡:18, 小鸡:78
公鸡:8, 母鸡:11, 小鸡:81
公鸡:12, 母鸡:4, 小鸡:84
运行次数: 1030301

```

还有更多的“优化代码”，请直接看代码文件！！

课间案例:

有红、白、黑三种球若干个，其中红、白球共 25 个，白、黑球共 31 个，红、黑球共 28 个，求这三种球各多少个？

6.8. 循环的中断

循环是按给定的条件，只要条件满足就会继续执行循环体的一种语法形式。

但，我们也可以在循环过程中（循环体内），人为将循环中断。

有两种中断循环的方式：

continue 中断：

含义：中断当前正在进行的循环体（即后续语句不再执行），继续下一次循环要执行的语句。

语法形式：

```
continue [ $n ];           //表示是要中断第几层的循环，继续该层循环的下一层。  
                           //其中 $n 可以省略，如果省略，表示 1，就是中断当前层的循环。
```

break 中断：

含义：停止（跳出）当前正在进行的循环（即完全终止循环），去执行该循环之后的语句。

语法形式：

```
break [ $n ];              //表示是要中断几层循环。  
                           //其中 $n 可以省略，如果省略，表示 1，就是中断当前循环
```

他们都适用于 3 种循环。

案例演示：

从 1-10（含 1 和 10）进行循环并输出该数字，并要求能被 3 整除就不输出，能被 9 整除就终止循环。
最后再输出循环变量的值。

7. 函数

7.1. 函数的概念与作用

函数不是数！

函数是一种代码形式（语法形式）。

函数是将“若干行代码”以一种语法形式包装成的一个整体。

该整体可以做到“需要的时候就去执行它”（就是执行其中的若干行代码）。

函数是解决在不同情形（不同代码位置）下需要执行相同代码的有效方式——即所谓**代码重用**。

函数通常用于“专业的事由专业的人来做”这种现实世界的常见现象。

比如油条摊：专门做油条，顾客随时需要，就随时可以买（付钱就可以）。

比如轮胎厂：专门生产不同型号轮胎，汽车制造厂随时需要就随时提供（付钱）。

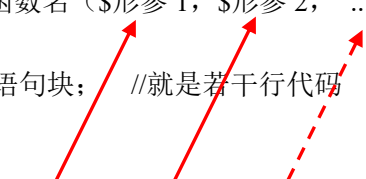
比如豆浆机：专门制作豆浆，人们随时需要豆浆就随时可以启动豆浆机制作豆浆（提供豆子和水）。



7.2. 函数的定义与调用

定义语法形式：

```
function 函数名 ($形参 1, $形参 2, ..... ) { //形参，就是形式参数，是变量  
    函数体语句块; //就是若干行代码  
}
```



调用语法形式：

```
函数名 ($实参 1, $实参 2, ..... ); //实参就是实际参数，是数据
```

说明：

- 1，函数名的命名规则，跟变量名一样；
- 2，定义函数的目的就是为了以后能够调用；
- 3，调用函数，其实就是执行函数中代码；
- 4，形参，其实就是变量，是只能在该函数内部使用的变量；
- 5，实参，其实就是数据，是会传入函数内部的数据（是一一对应地赋值给形参变量）；

案例演示：

写一个函数，可以给定一个半径，求对应圆面积。

课堂练习：

写一个函数，可以给定长宽高，求对应一个长方体的表面积。

7.3. 函数执行原理（重点/难点）

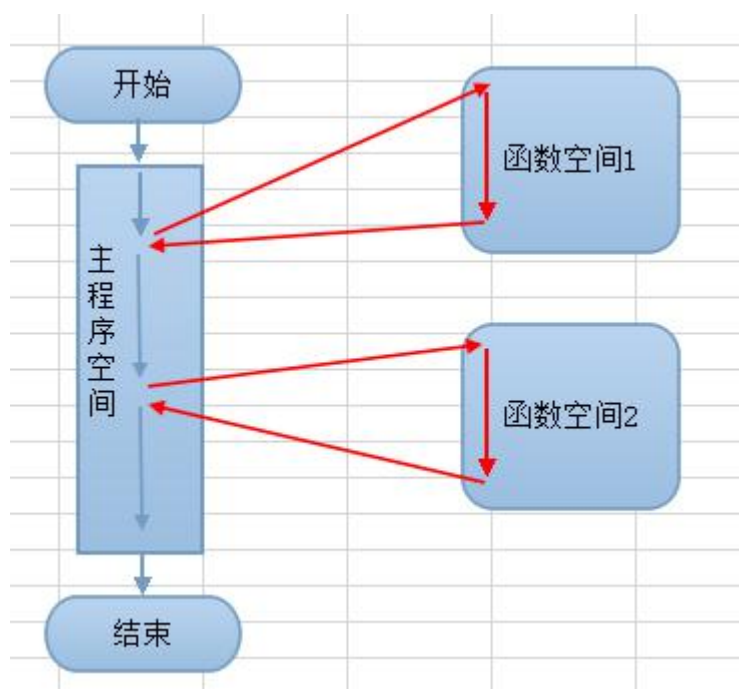
程序总是运行在一个“内存空间”。

程序开始执行的位置所在的空间，可以称之为“主运行空间”。

实际上，通常函数外面的那些程序，都是运行在主空间中。

那么，函数的运行，就相对独立了——每个函数的每次调用，都是运行在单独的一个自己的空间中。

如下所示：



7.4. 函数参数（重点）

7.4.1. 形参（形式参数）

就是定义函数的时候，在函数名后的小括号中给出的**变量名**。

形参，只能在函数内部使用——即该变量的使用范围仅仅局限于当前函数内部。

形参的本质是变量！

7.4.2. 实参（实际参数）

就是调用函数的时候，在函数名后的小括号中给出的**数据值**。

实参的本质是数据！

7.4.3. 函数参数的传值方式

含义：

实参变量的值，以什么方式传给形参。

说明：

其前提是：实参是一个变量的情况。

所以其实这里讨论的是：两个变量的传值方式问题。

默认情况下是值传递。



可以使用“&”符号设定为引用传递，形式如下：

```
function fl($p1, &$p2, .... ) {  
    . . . . .  
}
```

此时，在函数内部，对该形参变量改变其值，则对应的实参变量（在函数外部）的值也改变了。

案例演示：

7.4.4. 形参的默认值

形参可以设定默认值。形式为：\$形参名 = 某值。

设定默认值的形参，只能放在没有设定默认值的形参的后面（右边）。

设定了默认值的形参对应的实参可以不提供数据，此时函数就会使用该默认值当做实参的值。

```
function fl($p1, $p2, $p3 = 3, $p4 = true){  
    //函数体语句块  
}
```

则此时调用上述函数，以下形式都可以：

```
fl(1,2);  
fl(3, 4, 5);  
fl(6,7,8, 9);
```

案例演示：

定义一个函数，该函数可以计算给定半径的球的体积，其中圆周率 π 默认使用 3.14，也可以根据不同精度的需要给定不同的圆周率。

7.5. 函数返回值（重点）

含义：

一个函数在执行结束时，可以让其返回一个数据，这就是函数的返回值。

语法：

return 要返回的数据；

说明：

- 1，一般情况下一个函数执行结束都是需要返回一个数据值的。
- 2，函数也可以在执行的中途返回数据，此时，函数也是结束了的。
- 3，一个函数执行得到的返回值，可以在任何需要数据的场合使用，跟使用一个变量数据一样。

演示：

计算两数的平方和与两数的平方差相除的结果。



另外，`return` 语句也可以不带后面的数据，此时，就只是单纯地结束函数，并不返回数据（也可以说返回 `null` 这个空数据）。

演示：

将上述案例加强一个判断条件：当两个数相等时，报错并结束，不返回数据。

结论：

只要执行到 `return` 语句，函数就会结束，是否返回数据，看该行代码。

7.6. 可变函数

含义：

所谓可变函数，就是函数名是一个变量的情形。

可变函数实际上就是在调用函数的时候，使用一个变量来表示函数名，并用该变量去调用函数。

注意：定义函数的时候，不可以使用可变函数名！

对比：

可变变量，就是变量名是一个变量，比如：`$$v1`；

可变函数，就是函数名是一个变量，比如：

调用函数 `f1` 的语法是这样：`f1()`；

可以使用可变函数的语法来调用：`$func_name = "f1"; $func_name()`； //此时实际就是调用 `f1`

可变函数的本质是：一个变量的内容是一个字符串，该字符串是一个函数的名字，比如：

```
function f1(){ ..... }
```

```
function f2(){ ..... }
```

```
$f = "f1";
```

```
$f(); //调用了函数 f1，这就是可变函数！！
```

```
$f = "f2";
```

```
$f(); //调用了函数 f2，这就是可变函数！！
```

```
$f = "f3";
```

```
$f(); //报错！因为没有 f3 这个函数
```

7.7. 匿名函数

含义：

就是一个“定义时没有名字”的函数。

此时，就面临一个问题，那就是：没有名字，怎么调用呢？

实际上，此时它通过另一个方式来调用，如下所示：

```
$f1 = function (形参...) { ..... } //这是定义匿名函数的形式。  
$f1(实参); //这就是调用该调用。可见其调用，跟可变函数的写法非常类似。
```

案例：

定义一个匿名函数，该函数可以计算两个数的最小公倍数，并调用该函数算出 6 和 8 的最小公约数。

7.8. 系统常用函数介绍

PHP 语言，以函数极大丰富而闻名于世。

看手册，查手册，并作为一种学习的习惯：

7.8.1. 跟函数有关的函数

`function_exists(“函数名”)`：判断一个函数是否已经存在；

`func_get_arg($n)`：在函数内部可用，用于获得第 `n` 个实参（`n` 从 0 开始算起）

`func_get_args()`：在函数内部可用，用于获得所有实参，结果是一个数组

`func_num_args()`：在函数内部可用，用于获得实参的个数

上面 3 个函数，可以让我们在自定义的函数内部，直接访问（使用）实参数据，而不依赖于形参变量。这种特性，可以给我们定义某种“不确定有几个数据需要计算的”场合。

案例：

定义一个函数，该函数可以求出所给定的若干个数据中的奇数的和。

7.8.2. 字符串有关常用函数

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`,



7.8.3. 常用数学函数（重点）

max: 取得若干个数据中的最大值
min: 取得若干个数据中的最小值
round: 对某个数据进行四舍五入（可以设定保留几位小数）
ceil: 对某个数“向上取整”：将一个数据往上找出其小的一个整数（含其本身）。
floor: 对某个数“向下取整”：将一个数据往下找出其大的一个整数（含其本身）

```
$n1 = floor(4.1); //4
$n2 = floor(4.9); //4
$n3 = floor(4); //4
$n4 = floor(-4.1); //-5
```

abs: 取得某个数据的绝对值
sqrt: 计算某个数的开方值
pow: 对某个数进行“幂运算”（就是获得某个数的若干次方）

```
$n1 = pow(3, 2); //3 的 2 次方,9
$n2 = pow(2, 3); //8
$n3 = pow(1.5, 2); //2.25
$n4 = pow(1.5, 2.5); //。。。。1.5 的 2.5 次方
$n5 = pow(9, 0.5); //3, 就是开方, 相当于 sqrt()
```

rand: 获得某两个数之间的随机整数（含该两个数）
mt_rand: 获得某两个数之间的随机整数（含该两个数），比 rand 更快。

```
$n1 = mt_rand(0, 10); //随机数在 0-10 之间（含）
```

手册》函数参考》数学扩展》Math》Math 函数。

演示案例：

定义一个函数，该函数可以返回所给定的任意两个数字之间的随机整数。

7.8.4. 常用时间函数

time: 获得当前时间（精确到秒），结果其实一个“整数”而已，代表从 1970 年 1 月 1 日 0:0:0 秒到当前时刻的秒数。

microtime: 获得当前时间（可以精确到微秒）

mktime: 创建一个时间数据，参数为：时、分、秒，月、日、年

date: 将一个时间转换为某种字符串形式

idate: 取得一个时间的某个单项数据值，比如 idate(“Y”)取得年份数

strtotime: 将一个字符串“转换”为时间值；

date_default_timezone_set: 在代码中设置“时区”

date_default_timezone_get: 在代码中获取“时区”

案例：

课堂电脑性能大比拼：

计算从 1 加到 1000 万，看花了多少时间？

做法：先获得一个时间，然后计算，然后再获得一个时间，后一个时间，减前一个时间，就是耗时。

8. 函数相关

8.1. 变量的作用域问题（重点）

简单来说，有 3 种作用域：局部作用域，全局作用域，超全局作用域；

相对应的，有 3 种变量：局部变量，全局变量，超全局变量；

8.1.1. 局部作用域与局部变量：

就是函数内部范围的作用域，其中定义的变量就是局部变量（包括形参也是局部变量）。

局部变量只能在其所在的局部作用域中使用（访问）。

局部变量在函数调用结束时，会被自动销毁（可以理解为函数执行结束，该执行空间也被销毁了）。

```
$v1 = 1;
```

```
function fl( $p1 , $p2 ){    //此函数范围就是一个局部作用域，其中有 3 个局部变量：$p1, $p2, $v2
    $v2 = 2;
    echo $v1;    //报错！变量未定义
    echo $v2;
}
fl(); //
echo $v1;
echo $v2;    //报错！变量未定义
```

8.1.2. 静态变量：一个特殊的局部变量

含义：

在函数内部，使用 static 关键字修饰的变量。

形式：

```
function XXX( .... ){
    static $s1 = 10;    //此时，$s1 就是静态变量
    .....
```

```
}
```

静态变量的特点：

静态变量的值不会在函数调用结束时被销毁，而是会一直保留。

也就是说，当再次调用函数时，该变量（的值）还能继续使用。

对比演示：

一个普通局部变量，一个静态变量，都进行简单++操作。多次调用函数，并输出他们的值。

8.1.3. 全局作用域与全局变量：

就是函数外部范围的作用域，其中定义的变量就是全局变量。

全局变量只能在其所在的全局作用域中可以直接使用（访问）。

8.1.4. 超全局作用域与超全局变量：

包括局部作用域和全局作用域的整个作用域范围。

超全局变量可以在所有范围中使用（访问）。

实际上，只有有限的 10 来个系统预定义变量是超全局变量，包括：\$_GET, \$_POST, \$_REQUEST 等。

所以，系统预定义变量，也被统称为超全局变量。

PHP 中的不同作用域的图示：



一个特别的超全局变量：\$GLOBALS

它也是一个数组，其中存储了我们自己定义的所有全局变量。

每个全局变量的变量名，就是\$GLOBALS 数组的一个单元。

比如：

在全局作用域中定义如下变量：

```
$v1 = 1; //这一行执行，就有了一个这个：$GLOBALS['v1'], 其值为 1
```



```
$v2 = 'abc';      //这一行执行，就有了一个这个：$GLOBALS['v2'], 其值为'abc'
$v3 = true;       //这一行执行，就有了一个这个：$GLOBALS['v3'], 其值为 true
```

它可以让我们在局部作用域范围中，使用全局变量，方式如下：

```
$v1 = 10;        //全局变量
function func1( ){
    echo $GLOBALS['v1']; //输出 10;
    echo $v1;           //报错：变量 v1 未定义
    $s1 = $GLOBALS['v1'] * 5; //结果为 50;
    $s2 = $v1 * 5;       //报错：变量 v1 未定义
}
```

一个特别的的关键字：**global**

作用：

用于在局部作用域中，修饰一个跟全局变量同名的局部变量。

此时该局部变量也可以使用全局变量的值了——实际上他们其实是类似变量引用关系。

演示：

总结：

`$GLOBALS` 数组（变量）和 `global` 关键字，都能实现：在局部作用域中使用全局变量。

8.2. 递归函数（重点/难点）

基本含义：

就是一个函数内部再调用该函数本身的一种情形，这是语法形式上的。

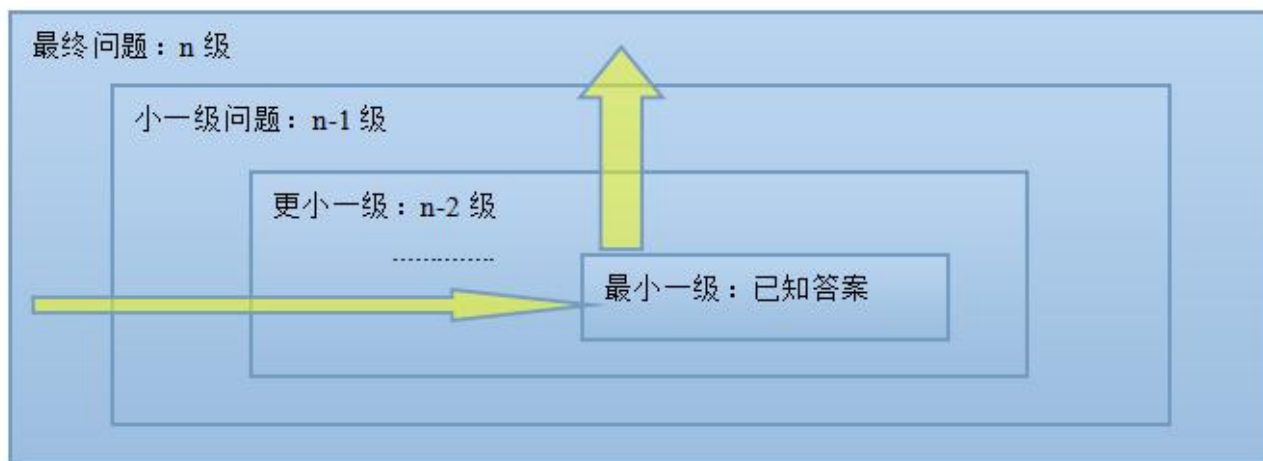
具体场景是：

如果要解决的“最终问题”，可以根据比该问题“小一级”的问题的答案而得到解决，并且，该“小一级”的问题，还可以根据比其“更小一级”的问题的答案而得到解决，以此类推，直到“最小一级”的问题。如果最小一级问题已知，则最终的问题也就解决了。

危险：

如果函数在执行的过程中没有一个“不再调用”的终结机制，那么就会出现“停不下来”的现象。

原理：



递归调用过程的代码演示：

分析一下代码的输出结果：

```
function fl($n){  
    $n++;  
    echo "$n ";  
    if($n < 5){  
        fl($n);  
    }  
    echo "$n ";  
}  
fl(1);
```

案例 1：

计算 5 的阶层：

分析：

数学上阶乘可以这样来描述：一个数 n 的阶乘，是 $n-1$ 的阶乘，乘以 n 的结果！

假设，我们有一个函数 jiecheng(),它可以计算任意正整数 n 的阶乘，类似这样：

$n = 5$; //或等于 10, 13, 等等都无所谓。

$result = jiecheng(n)$;

案例 2：

计算斐波那契数列第 10 项的值：1, 1, 2, 3, 5, 8, 13, 21,

假设有个函数，可以计算斐波那契数列的第 n 项：



9. 文件加载

9.1. 文件加载的含义

含义：

将一个（别的）文件包含到当前文件中，成为当前文件运行过程中的一部分。

通常，一些公共的代码，在多个页面都需要用到的时候，会做成一个独立的文件。

然后在不同的页面需要用到时候，直接载入进来就可以了。

语法：

`include “要载入的文件路径”;` //可以是相对路径，或本地物理路径。

说明：

可以载入 `php` 文件，也可以载入 `html` 文件。

示例：

原理：

载入一个文件的本质是：将被载入的文件“插入”到当前载入代码所在的位置。

9.2. 文件加载的四种方式（重点）

四种方式如下所示：

`include ‘要加载的文件’;`

`include_once ‘要加载的文件’;`

`require ‘要加载的文件’;`

`require_once ‘要加载的文件’;`

其中，“要加载的文件”，是一个文件路径，可以是相对路径，也可以是物理路径，或直接文件名：

相对路径：

`‘./文件名’;`

`‘./dir1/文件名’;`

`‘./dir1/dir2/文件名’;`

`‘../文件名’;`

`‘../dir1/文件名’;`

`‘.././dir1/dir2/文件名’;`

等等。。。

物理路径：

`‘c:/itcast/class/php66/day4/文件名’;` //window 系统



‘path1/path2/文件名’; //linux、unix 等系统
等等。。。

获取物理路径（绝对路径）的方式：

DIR：表示当前文件所在路径，由它可以构建出绝对路径；

getcwd()：表示当前正访问的网页路径，由它也可以构建出绝对路径；

非相对非绝对路径（其实就是没有给出路径，只给出文件名）：不推荐！

形式为：`include ‘文件名’;`

此时，会按如下顺序去寻找该文件：

- 1，先在 `php.ini` 中 `include_path` 项设定的目录中寻找该文件；
- 2，如果上一步没有找到，就在当前工作目录（由 `getcwd()` 获取）下寻找该文件；
- 3，如果上一步没有找到，就在当前载入语句的文件所在目录（由 `__DIR__` 获取）下寻找；
- 4，如果上一步还是没有找到，就报错了。

9.3. 四种方式的区别

`include`：每次都载入文件（可能会重复载入），如果载入失败，在报错后继续执行后续语句；

`include_once`：只载入一次（不会重复载入），如果载入失败，在报错后继续执行后续语句；

`require`：每次都载入文件（可能会重复载入），如果载入失败，在报错后终止程序；

`require_once`：只载入一次（不会重复载入），如果载入失败，在报错后终止程序；

一般来说，如果被载入的文件内容，是后续代码运行的必备前提，则应该使用 `require` 载入。

如果被载入的文件内容，只需要（或只允许）出现一次，则应该使用“`xxxx_once`”载入。

10. 错误处理

10.1. 错误分类

语法错误：

程序不能运行，是在运行之前，检查语法的时候，就发现语法出错，结果是提示错误，不运行程序。

运行时错误：

语法检查没错，然后开始运行，在运行中出现了错误，然后报错。

这是开发中最常见的错误。

逻辑错误：

程序能运行，且一直到结束没有报错，但执行得到的结果却是错的。

10.2. 常见错误代号（重点）

含义：

是指在程序运行时，发生的错误，系统会针对每种错误，给出相应的错误代号，并进行提示（报错）。
另外，程序如果在运行之前检查语法的时候就发现语法错误，也会报错，也有一个错误代号。

常见错误代号有：

E_NOTICE:

提示性错误，轻微；
错误发生后，后面的程序继续执行。

E_WARNING:

警告性错误，稍微严重；
错误发生后，后面的程序继续执行。

E_ERROR:

严重错误/致命错误；
错误发生后，后面的程序不再执行！

E_PARSE:

语法错误（语法解析错误）；
语法解释错误，是直接就不运行程序。

E_USER_NOTICE:

用户自定义的提示错误

E_USER_WARNING:

用户自定义的警告错误

E_USER_ERROR:

用户自定义的严重错误

E_ALL:

它是一个代表“所有”错误的代号。

说明：

- 1，这些错误代号，其实只是系统预先设定的一些常量，他们的值大约是：1， 2， 4， 8， 16.....
- 2，这些错误代号，通常只是用于对错误控制时进行“设置”使用。
- 3，他们是一系列的整数，并具有一定的规律：1,2,4,8,16,32,64，。。。。
- 4，可以在 php.ini 中使用（设置）他们，如下所示：



```

425 ;
426 ; Error Level Constants:
427 ; E_ALL - All errors and warnings (
428 ; E_ERROR - fatal run-time errors
429 ; E_RECOVERABLE E_ERROR - almost fatal run-time
430 ; E_WARNING - run-time warnings (non-fa
431 ; E_PARSE - compile-time parse errors
432 ; E_NOTICE - run-time notices (these a
433 ; from a bug in your code,
434 ; intentional (e.g., using
435 ; relying on the fact it's
436 ; empty string)
437 ; E_STRICT - run-time notices, enable

```

10.3. 错误触发

就是发生了一个错误的意义——即触发了错误。

有两种情形会触发错误：

- 1，程序本身有错，则运行时就会触发错误（并提示）。
- 2，程序本身没错，但出现不符合预计的情形（比如数据不符合要求）。
此时程序员可以主动触发一个错误，也可以说是由程序员“主动创建一个错误”——这就是“用户错误”，包括：

E_USER_NOTICE:
E_USER_WARNING:
E_USER_ERROR:

如何触发“用户错误”呢？

自定义错误触发语法：

`trigger_error(“自定义错误提示内容”， 自定义错误的代号);`

案例演示：

10.4. 错误显示设置

如果有错误发生（触发了错误），默认情况下会被显示在页面（即输出的结果页面）。



我们可以对此进行设置，以决定以下两点：

- 1，设置 `display_errors` 以决定是否显示错误：

在 `php.ini` 中设置：`display_errors = On` 或 `Off`

这里设置，影响所有使用该 `php` 语言引擎的代码（网站页面）；

在 `php` 文件中设置：`ini_set('display_errors', 1 或 0);` `//1` 表示显示，`0` 不显示

在这里设置，只影响当前网页代码本身。

- 2，设置 `error_reporting` 以决定显示哪些错误：

在 `php.ini` 中设置：`error_reporting = 错误代号 1 | 错误代号 2 |`

`//(要显示的就写出来，或者可以写 E_ALL，表示显示所有)`

在代 `php` 文件中，道理类似：`ini_set('error_reporting', 错误代号 1 | 错误代号 2 |`

代码演示：

10.5. 错误日志设置

如果有错误发生（触发了错误），默认情况下不会将错误信息记录（保存）下来。

我们可以对此进行设置，以决定以下两点：

- 1，设置 `log_errors` 以决定是否记录错误：

`php.ini` 中设置：`log_errors = On` 或 `Off`

代码文件中设置：`ini_set('log_errors', 1 或 0)`

- 2，设置 `error_log` 以决定记录到哪里：

通常，就设置为一个文件名，`php` 系统会在网站的每个文件夹下都建立该文件，并记录错误。

`php.ini` 中：`error_log = error.txt;` `//它是纯文本的`

代码中：`ini_set("error_log", 'error.txt');`

演示代码：

10.6. 自定义错误处理（重点/难点）

之前，我们面对的情形都是错误发生的时候，系统生成错误，并处理错误（给出错误信息）。

我们能控制的就只是：是否显示，显示什么，是否记录，记录到哪里？

实际上，我们也可以更进一步控制错误信息，以决定错误发生的时候，显示什么样的错误信息。

这就是“自定义错误处理”。

具体做法，分 2 步：

第 1 步：

声明错误发生时，由我们自己来处理——设定一个错误处理的函数名。

第2步：

定义该函数，在函数中详细设定错误的处理情况：怎么显示，显示什么，怎么记录，记录什么。

演示案例：

运算符详解、计算机码介绍

文档目录：

一、 字符串详解.....	85
(一) 4 种不同形式的字符串.....	85
1. 单引号字符串.....	85
2. 双引号字符串.....	85
3. heredoc 字符串.....	86
4. nowdoc 字符串.....	86
(二) 转义字符.....	87
(三) 字符串的长度问题.....	87
(四) 常用字符串函数（重点）.....	88
二、 数组详解.....	90
(一) 数组的概念和定义.....	90
(二) 数组下标问题.....	90
1. 下标的可用值.....	90
2. 整数下标的特性.....	91
3. 索引数组.....	91
4. 关联数组.....	91
(三) PHP 数组的维数.....	91
(四) 数组的遍历（重点）.....	92
1. 使用 foreach 语句遍历数组.....	92
2. 使用 for 循环语句遍历数组.....	93
(五) 常用数组函数.....	93
(六) 数组排序算法（重点/难点）.....	94
1. 数组的排序问题.....	94
2. 冒泡排序算法.....	94
3. 选择排序算法.....	95
(七) 数组查找算法.....	95
1. 二分查找：.....	96
三、 文件加载.....	错误！未定义书签。
(一) 文件加载的含义.....	错误！未定义书签。
(二) 文件加载的四种方式（重点）.....	错误！未定义书签。

(三) 四种方式的区别.....错误！未定义书签。

今日主要内容

字符串：

掌握 4 种不同形式的字符串的定义与区别；

理解字符串的长度问题；

掌握常用的转义字符；

认识一些常用的字符串函数；

数组：

掌握数组的基本概念和基本使用；

理解 PHP 数组的元素顺序问题；

能够使用 foreach 遍历任意数组；

掌握数组的指针函数并使用 for 循环遍历任意数组；

掌握两个数组排序算法

理解数组的查找算法

11. 字符串详解

11.1. 4 种不同形式的字符串

11.1.1. 单引号字符串

形式： `$s1 = '字符串内容'`

特点：

只能使用 2 个转义符： `\\`（代表一个反斜杠） `\'`（代表一个单引号）

通常，如果没有其他特殊需求（比如字符串中使用一些转义符，以及一些变量），则推荐使用单引号字符串。！

11.1.2. 双引号字符串

形式： `$s1 = "字符串内容"`

特点：

1，能使用较为丰富的转义符，包括：`\\` `\'` `\n` `\r` `\t` `\$`等

`\n` ：代表“换行符”（就是一个新行）

`\r` ：代表“回车符”（其实也是一个新行）

`\t` ：代表“tab 符”，



`\$`：代表“\$”本身，因为双引号字符串中能识别（解析）变量，则如果不想去解析，就用此转义！

```
$v1 = 10;
```

```
echo “结果为: $v1”; //输出内容为: 结果为: 10
```

```
echo “结果为: \$v1”; //输出内容为: 结果为: $v1
```

2, “\$”符号在其中会被识别为是变量的起始符号，并试图读取变量值——即能识别变量；

3, 识别其中变量，建议使用大括号括起来，类似这样：{\$变量名}， {\$数组[‘下标’]}

11.1.3. heredoc 字符串

形式：\$s1 = <<< “标识符”

这里写字符内容，可以多行写

标识符;

特点:

1, 特点跟双引号字符串一样！

注意:

标识符结束那一行，只能出现标识符及紧挨着的分号，任何其他字符都不可以出现。

11.1.4. nowdoc 字符串

形式：\$s1 = <<< ‘标识符’

这里写字符内容，可以多行写

标识符;

特点：无特点，是最“纯净”的字符串，写什么就是什么。

heredoc 字符串，和 nowdoc 字符串，适用于表达（描述）一大段内容的字符串，特别是适合于写 html 部分的代码（含 js，css 等）。

如下：



```

7 <body>
8 <?php
9
10 $js1 = <<<"script"
11 <script>
12     var v1 = 1 + 2;
13     alert(v1);
14 </script>
15 script;
16
17 $html1 = <<<'HTM1'
18     <h1>标题</h1>
19     <input />
20     <a href='http://www.baidu.com' >百度</a>
21 HTM1;
22
23 echo $js1;
24 echo $html1;
25

```

11.2. 转义字符

转义的字符的本质，其实是在一个字符串的语法形式中，如何来表达一些相对特殊的一些字符的问题。比如：

双引号字符串：\$s = “这里字符串内容，要是直接出现双引号就会有语法问题。”;

11.3. 字符串的长度问题

字符串的长度问题，有两个方面的理解：

- 1，一个字符串有几个字符（人可见到的字符个数）；
- 2，一个字符串占据多少个字节空间（人不可见）；

几个常识：

- a, 1 字节（B）就是 8 个 bit 位（最小的存储空间），1KB=1024B，1MB=1024KB，1GB=1024MB。。
- b, 一个英文字符占据 1 字节空间，gbk 编码中 1 个汉字占据 2 个字节，utf8 编码 1 个汉字占 3 字节。

求 php 字符串的长度，，有两个函数：

strlen(字符串)：

求该字符串的“字节数”，也就是占据的字节空间大小；

mb_strlen(字符串)：



求该字符串的“字符个数”。

```

9  $s1 = 'abc';
10 $len11 = strlen($s1); //取得字节数，对英文，字节数，也是字符数
11 echo "<br />s1的字节数为: $len11";
12 //echo "<br />s1的字符数为: $len1";
13
14 $s2 = 'abc中英混合';
15 $len21 = strlen($s2); //取得字节数，
16 $len22 = mb_strlen($s2); //取得字符数
17
18 ?>
19 </body>
20 </html>

```

s1的字节数为：3
Fatal error: Call to undefined function mb_strlen() in H:\i
需要到php.ini中开启一个模块：mb_string

该模块准确的位置如下：

查看(V) 转到(G) 工具(I) 项目(P) 自选项(N) 帮助(H)

```

876 ;extension=pnp_interbase.dll
877 ;extension=php_ldap.dll
878 ;extension=php_mbstring.dll
879 ;extension=php_exif.dll ; Must be after
880 extension=php_mysql.dll

```

去掉前面的分号，重启apache

```

9  $s1 = 'abc';
10 $len11 = strlen($s1); //取得字节数，对英文，字节数，也是字符数
11 $len12 = mb_strlen($s1, "utf-8"); //取得字符数
12 echo "<br />s1的字节数为: $len11";
13 echo "<br />s1的字符数为: $len12";
14
15 $s2 = 'abc中英混合';
16 $len21 = strlen($s2); //取得字节数，
17 $len22 = mb_strlen($s2, "utf-8"); //取得字符数
18 echo "<br />s2的字节数为: $len21";
19 echo "<br />s2的字符数为: $len22";
20

```

s1的字节数为：3
s1的字符数为：3
s2的字节数为：15
s2的字符数为：7

11.4. 常用字符串函数（重点）

字符串输出：

- echo： 输出一个或多个字符（不是函数，是语言结构）
- print： 输出一个字符串
- print_r： 输出变量的较为详细的信息
- var_dump： 输出变量的完整信息

字符串去除与填充：

- trim： 消除一个字符串两端的空白字符或指定字符（空白字符包括：空格，\n, \r, \t 等）
- ltrim： 消除一个字符串左边的空白字符或指定字符



第 89 页



12. 数组详解

12.1. 数组的概念和定义

数组，是指将若干数据按一定的顺序组合为一个整体。

每个数据被称为一个“单元”——数组单元。

每个单元由两部分构成：下标和值，下标也称为“键”（key），

数组的一个重要特点是：其中的数据有明确的顺序，而是，是其放入数组时的先后顺序。

数组有如下几种定义形式：

形式 1：

`$arr1 = array(单元 1, 单元 2, ...);`

形式 2：

`$arr2 = [单元 1, 单元 2, ...];`

单元（元素）的形式为： `[下标=>]值`

形式 3：（不推荐）

`$arr3[下标 1] = 值 1;`

`$arr3[下标 2] = 值 2;`

.....

数据取值的语法形式：

`$数组名[下标];` //其中，下标可以是整数的，也可以是字符串的（注意有引号）

12.2. 数组下标问题

12.2.1. 下标的可用值

可以使用整数或字符串。



12.2.2. 整数下标的特性

可以使用任意整数，也可以不显式使用下标，此时默认就是整数下标。

而且，从前往后，每一个没有使用下标的单元，系统给其分配的下标为之前所用过的整数下标的最大值+1（对于第一个是0）。

```
$arr1 = array('a', 2=>'b', 'c', 'x'=>'d', 'e'); // 其下标分别为: 0, 2, 3, 'x', 4
$arr2 = array(5=>'a', 2=>'b', 'c', 'x'=>'d'); // 其下标分别为: 5, 2, 6, 'x'
$arr3['x'] = 5; //这一行，会自动创建一个数组，
$arr3[] = 6; //此时下标就是 0
```

12.2.3. 索引数组

通常是指一个数组的下标是从0开始的连续的整数。

举例 1:

```
$arr1 = array(5, 8, 12, 2, 3);
$arr2 = [8, 22, 24, 22, 12];
$arr3[] = 8;
$arr3[] = 24;
$arr3[] = 22;
```

12.2.4. 关联数组

通常是指一个数组的下标都是字符串。

```
$person = array(
    'name'=>'张三',
    'age'=>18,
    'edu'=>'大学',
    'salary'=>10000,
    'from'=>'北京',
);
```

12.3. PHP 数组的维数

按通常的数组元素的复杂程度，数组可以分为一维数组，二维数组，三维数组等等。

一维数组:

数组的每一个单元的值都是一个“非数组”值。

```
$arr1 = array(11, 12, 13, 14);
```

二维数组:

数组的每个单元的值都是一个“一维数组”。

```
$arr2 = array(
    array(11, 12, 13),
```

```
array(21, 22, 23),  
.....  
);
```

三维数组：

数组的每个单元的值都是一个“二维数组”。

多维数组：

依此类推。。。。

不整齐数组（异形数组）：

实际上，由于 PHP 的数组值可以是“任意数据”，因此，PHP 数组的维数其实没有太大实际意义的。

所谓维数，其实是另一些编程语言中的数组的“整齐”格式的说法：一维数组类似排成一排的格子（线）；二维数组类似排成一个平面的格子（面）；三维数组类似堆满了一屋子的格子（体）。

而 php 数组，却可以更为灵活，类似这样：

```
array(  
    1, 2, array(31, 32, ), 4,  
    array(51, 52, 53, array(541, 542, 543, 544)),  
    6, array(71, 72, 73),  
);
```

这种数组就不好说几维的了，而可以称为“异形数组”。

12.4. 数组的遍历（重点）

12.4.1. 使用 foreach 语句遍历数组

遍历：就是对数组的每一项都“访问”到并进行所需要的数据处理。

```
foreach( $数组名 as [$key =>] $value){  
    //这里，$key 和$value 只是变量，它会在遍历数组的过程中，按顺序依次取得数组每个单元的下标和值。  
    echo "<br />{$key} >>> {$value} “;  
}
```

演示案例 1：

输出以下这个数组的每一项，听求其平均值：

```
$arr1 = array(11, 12, 13, 14);
```

演示案例 2：

使用数组的遍历语法，求以下这个数组的最大值及其下标！

```
$arr2 = array(11, 18, 21, 14, 8);
```




12.4.2. 使用 for 循环语句遍历数组

数组的指针：每一个数组内部，都有一个“指针”，正常情况下，指针指向数组的某个单元，起初默认是指向第一个单元。

对于数组 `$arr1 = array(18, 22, 13, 28, 15, 33,);`

图示如下：



下标	0	1	2	3	4	5
值	18	22	13	28	15	33

初始状态下，指针指向数组的第一个单元。

php 中，有如下几个函数，可以针对数组指针进行相应操作：

```
$re = current( $arr1);    //取得数组中当前指针所在单元的值；
$re = key( $arr1 );       //取得数组中当前指针所在单元的键（下标）；
$re = next( $arr1 );      //将数组中的指针往后移动一个位置，并取得新位置上的值；
$re = prev( $arr1 );      //将数组中的指针往前移动一个位置，并取得新位置上的值；
$re = end( $arr1 );       //将数组中的指针移动到最后一个位置，并取得新位置上的值；
$re = reset($arr1);       //将数组中的指针移动到最前一个位置，并取得新位置上的值；
```

课堂练习：

请用 for（即不用 foreach）来遍历如下数组，并按顺序输出其每一个单元的键和值：

```
$arr1 = array('a', 2=>'b', 'c', 'x'=>'d', 'e'); //提示：count()函数可以求数组的长度
```

12.5. 常用数组函数

手册》函数参考》变量与类型相关扩展》数组》数组函数

`max()`: 获取一个数组中的最大值

`min()`: 获取一个数组中的最小值

`count()`: 获取一个数组的元素个数

`in_array()`: 判断一个数据是否在指定数组中。

语法形式：`$b = in_array($数组, 数据);` //结果 true 或 false

`range()`: 生成某个范围的连续值的数组，比如 `range(3, 9)`会得到数组：`array(3, 4, 5, 6, 7, 8, 9,);`

`array_keys()`: 取出一个数组中的所有“键”并放入一个索引数组中。

`array_values()`: 取出一个数组中的所有“值”并放入一个索引数组中。

`array_push()`: 将一个或多个数据放入一个数组的“末端”。

`array_pop()`: 将一个数组的最后一个单元删除，并返回该单元的值。

`array_reverse()`: 将一个数组的所有单元的顺序进行反转（最前的放最后，最后的放最前）

12.6. 数组排序算法（重点/难点）

12.6.1. 数组的排序问题

常用的排序函数：sort, rsort, asort, arsort

数组排序函数大全：

排序函数属性				
函数名称	排序依据	数组索引键保持	排序的顺序	相关函数
array_multisort()	值	键值关联的保持，数字类型的不保持	第一个数组或者由选项指定	array_walk()
asort()	值	是	由低到高	arsort()
arsort()	值	是	由高到低	asort()
krsort()	键	是	由高到低	ksort()
ksort()	键	是	由低到高	asort()
natcasesort()	值	是	自然排序，大小写不敏感	natsort()
natsort()	值	是	自然排序	natcasesort()
rsort()	值	否	由高到低	sort()
shuffle()	值	否	随机	array_rand()
sort()	值	否	由低到高	rsort()
uasort()	值	是	由用户定义	uksort()
uksort()	键	是	由用户定义	uasort()
usort()	值	否	由用户定义	uasort()

12.6.2. 冒泡排序算法

原理：

遍历一个数组，在此过程中，将相邻的两个单元的值进行比较：如果前面的比后面的大，则将两个值交换位置。这个过程到最后，数组中的最大值一定放在最后位置了。

如果将上述过程再进行一遍，则又可以确定剩余数据中的最大值放在倒数第二的位置。

然后将上述过程继续进行一遍，则可以继续确定剩余数据中的最大值放在倒数第三的位置。

依次类推。。。。。进行若干次，就排好了。

图示：

有数组：\$arr1 = array(18, 22, 12, 15, 23, 9);

原始数组	18	22	12	15	23	9
第 1 趟之后						
第 2 趟之后						
第 3 趟之后						
第 4 趟之后						
第 5 趟之后						

规律总结：

代码演示：

12.6.3. 选择排序算法

原理：

遍历一个数组，在此过程中，找出数组中的最大值及其位置。然后将该最大值的单元，跟数组的最后一个单元“交换位置”，这样进行一趟，数组中的最大值就一定放在最后位置了。

将上述过程中剩余的数据继续遍历一次，并做同样的事情，则此时剩余部分的最大值也能够放在剩余部分的最后位置——对整体而言就是倒数第二的位置。

依此类推。。。。。。进行若干次，就排好了。

图示：

有数组：\$arr1 = array(18, 22, 12, 15, 23, 9);

原始数组	18	22	12	15	23	9
第 1 趟之后						
第 2 趟之后						
第 3 趟之后						
第 4 趟之后						
第 5 趟之后						

规律总结：

代码演示：

结果：

```
交换前：Array ( [0] => 18 [1] => 22 [2] => 12 [3] => 15 [4] => 23 [5] => 9 )
交换后：Array ( [0] => 9 [1] => 12 [2] => 15 [3] => 18 [4] => 22 [5] => 23 )
```

12.7. 数组查找算法

查找算法，就是从一个数组中，找一个“目标”（可以是值，也可以是键）的算法。

数组的查找函数：

- in_array() : 在一个数组中找是否存在某个数据。
- array_search() : 在数组中搜索给定的值，如果成功则返回相应的键名
- array_key_exists() : 检查给定的键名或索引是否存在于数组中

一种查找算法：

遍历查找：不值得一提！

12.7.1. 二分查找：

此查找算法针对的数组有以下条件：

- 1，索引数组；
- 2，数组是已经排好序的了。

算法演示：