第2章 PHP基础语法

2.1 PHP标记与注释

目标：PHP语法规则是为掌握PHP开发的基础规范，为后续学习和开发做准备

- PHP标记：PHP代码起始标记
- PHP注释：描述代码逻辑
- PHP语句结束符：一行代码的结束标记

2.1.1 PHP标记

提问：PHP可以嵌入到HTML中，那么PHP引擎如何准备识别PHP代码并进行解析呢?

答案：PHP标记

目标：掌握PHP的标记应用

概念：

PHP标记：标记是用来帮助PHP引擎识别PHP代码的，PHP标记分为两个部分

标记开始：<?php

标记结束：?>

步骤

1、在开始编写PHP代码之前先使用开始标记：<?php

2、开始标记之后的所有内容都属于PHP引擎解析的范畴

3、在PHP代码写完之后使用结束标记：?>

4、如果PHP代码是一个独立的PHP文件（或者一直到最后都是PHP代码），那么可以没有结束标记?>

示例：

1、有结束标记的使用（多用于嵌入到HTML中）

```
<html>
    <head>
    </head>
    <body>
        <!--开始PHP代码书写：先写出PHP开始标记<?php-->
        <?php
            这里的内容都属于PHP的，PHP引擎也只会解析这部分内容
        ?>
        <!--PHP代码书写结束，需要使用PHP结束标记?>-->
    </body>
</html>
```

2、没有结束标记的使用（多为独立的PHP文件）

<?php

都是结束标记（包括空行到结尾）

PHP引擎会自动解析到最后

例1：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <?php
        echo 'hello world';
</body>
</html>
```

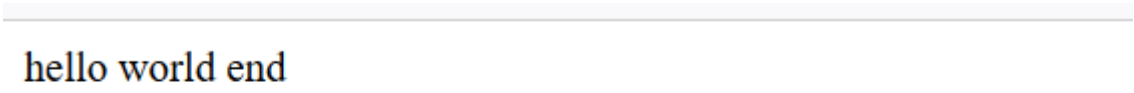
运行结果：

Parse error: syntax error, unexpected '<', expecting end of file in **D:\phpstudy_pro\WWW\myphp\common\1.php** on line **12**

例2：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <?php
        echo 'hello world';
    ?>
</body>
</html>
<?php
    echo 'end';
```

运行结果：



小结

1、PHP标记是为了让PHP引擎能够识别PHP代码

2、PHP标记分为起始和结束两个部分

- 起始标记：<?php ， 是PHP代码开始前必须先写好的

- 结束标记：?>,如果是独立PHP文件，不建议使用；如果是嵌入在HTML，最好使用。

2.1.2 PHP注释

提问：写代码时，如何清晰的让自己或者团队知道代码有什么作用?

答：注释

目标：掌握注释的实际应用

概念

注释：注释是一种使用代码符号之外的、通俗易懂的文字描述代码作用多逻辑的符号。PHP引擎在解析时，会忽略注释标记及被注释的内容。

注释分为两种：

行注释：#或者//，标记当前行符号后面的内容

块注释：/* ... */，标记起始和结束符号内的所有内容

使用规则

1、行注释：当前注释描述的内容不多，或者不会超过一行

- 独占一行使用
- 在代码之后使用

例：

```
<?php
#当前脚本是用来练习注释的
echo 'hello world';  //输出
```

2、块注释：当前注释描述的内容较多，通常会有多种信息或者内容较多超过一行。

- 独占一行开始：/*
- 独占一行结束： */

例：

```
/*
  从这里开始全是注释
 *@作者：小爱
 *@功能：注释
*/
```

小结

1、注释是为了让人在看代码时能够容易看懂代码

2、注释分为两种基本类型

行注释：使用#或者//，用来注释简单描述

块注释：使用/* .... */，用来注释复杂描述

3、一个具有良好职业素养的程序员应该尽可能多的使用注释来描述自己的代码

2.1.3 语句结束符

提问：PHP写代码的时候，系统怎么能区分代码是一部分还是一个整体呢?

回答：语句结束符

目标：掌握语句结束符在代码开发中的使用

概念

语句结束符：是一种必须的，显示的书写出来，用来告知PHP引擎，当前代码是一个整体的符号。在PHP中必须使用；（英文符号）来作为一行代码的语句结束符

示例：

```
<?php
#下面是一行独立代码（√）
echo 'hello world';  #echo 'hello world'是一个整体，而且结束了，所以必须使用；
#下面是一行错误独立代码（×）
echo 'Hi girl'       #没有语句结束符;系统会认为还没有结束，会报错
```

运行结果：

Parse error: syntax error, unexpected end of file, expecting ',' or ';' in **D:\phpstudy_pro\WWW\myphp\common\2.php** on line **5**

小结

1、PHP语句结束符是用来表明一行代码已经书写完毕

2、PHP语句结束符使用;

3、PHP语句结束符必须存在，否则报错

4、养成一个写完代码就写语句结束符的习惯，让代码少小错误。

小结

1、PHP基础知识包含三个内容

- PHP标记：让PHP引擎识别PHP代码

  起始标记（必写）:<?PHP

  结束标记（看情况）：?>
- PHP注释：用通俗的文字描述代码的作用

  行注释：注释一行信息，使用#或者//

  块注释：注释多行信息，使用/* ... */

  合格的程序员要养成良好的注释习惯
- 语句结束符：让PHP引擎知道一行代码已经写完

  使用方式: ;(英文分号)

2、掌握了PHP语法规则，我们就可以开始真正学习PHP的内在知识

2.2 PHP基础知识

目标：掌握PHP基本的知识点，运用基础知识实现PHP代码的数据加工处理

变量：变量的作用和操作

常量：常量的作用和操作

PHP数据类型：数据类型的概念和应用

PHP运算符：运算符的作用和应用

2.2.1 变量

提问：编程语言一定会涉及到数据加工，那么数据在程序中是如何保存的呢？

答案：变量

目标：掌握PHP中变量的定义和使用

概念：

PHP变量：是一种能过存储数据的符号

所有的变量都是以 $ 符号开始

变量所保存的数据都可以随意改变

变量必须先定义才能使用，否则报错

步骤：

1、需要定义变量

2、访问变量

3、修改变量

4、删除变量

示例：

```
#定义变量：将数据值10赋值给变量$num
$num=10;

#访问变量：输出变量的值
echo $num;

#修改变量的值：将10转成hello world
$num='hello world';

#删除变量
unset($num);
echo $num;
```

变量规则

1、PHP变量以$符号开始

2、变量的名字是由字母、下划线和数字构成，其中不能以数字开头

3、PHP中变量名是区分大小写的，所以a和A是两个变量

4、变量命名应该见名知意，如名字name，年龄age

5、复杂变量名可以使用驼峰法（serverName）或者下划线法（server_name），推荐下划线法

小结

1、PHP中变量以$符号开始

2、变量是用来保存一些可以变化的数据的

3、变量需要先定义才能访问，否则报错：未定义的变量

4、变量的操作包含增查改删

5、PHP是一种弱类型语言，所以变量可以存储任何数据

6、变量名区分大小写

7、变量名字应该见名知意

2.2.2 可变变量

可变变量：如果一个变量保存的值刚好是另外一个变量的名字，name可以直接通过访问一个变量得到另外一个变量的值：在变量前面再多加一个$符号。

```
<?php
$a='say';
$say='Hello';
$Hello='Lucy';
echo $a;
echo '<br>';
echo $$a;
echo '<br>';
echo $$$a;
?>
```

2.2.3 常量

提问：变量可以变化，如果有些数据不希望被乱改怎么办?

目标：掌握常量的概念以及PHP中常量的使用

概念

常量：一种存储数据的符号

- 常量通常大写字母
- 常量数据一旦定义，不可改变

步骤

1、定义常量

使用函数：define('常量名'，常量值);

使用关键字：const 常量名=常量值;

2、访问常量

直接访问：常量名

使用函数：constant（'常量名');

示例：

```
#使用函数定义
define('VERSION',1.0);
#使用关键字定义
const PI=3.14;

#直接访问
echo PI;
#使用函数访问
echo constant('VERSION');

const PI=5;//不允许修改
```

常量规则

1、常量名字通常使用大写（允许小写，但是程序员默认都使用大写）

2、名字由字母、下划线和数字组成，不能以数字开头

3、常量名定义应该做到见名知意

4、复杂常量名通常使用下划线法

5、常量名字可以比较松散，但是不建议使用，如define（'-_-','笑脸'）是系统允许的（访问时就必须使用函数访问）

扩展

1、PHP7以后常量允许定义数据常量（一次性定义多个常量），访问时使用数组下标访问

```
#定义数组常量
const MATH=array(
    'PI'=>3.14,
    'E'=>2.72
);
#访问
echo MATH['PI'];
```

小结

1、常量是用来保留一些不希望改变的数据

2、常量使用函数define或者关键字const定义，二者有些区别，在后面学习的时候会遇到

3、常量在定义后只能访问，不能修改和删除

4、PHP常量在PHP7以后允许值为数组

2.2.4 系统常量

提问：系统内部有没有像预定义变量那样的预定义常量呢?

回答：有，系统常量

目标：了解PHP中常见的系统常量，掌握常用几个使用方法

概念

系统常量：系统内部定义的常量，用户可以直接访问，也只能访问

- 固定系统常量：值是固定的
- 魔术常量：值是可变的

常用系统常量

固定系统常量

常量名	功能描述
PHP_VERSION	获取PHP的版本信息，如7.3.19
PHP_OS	获取解析PHP的操作系统类型，如：WINNT
PHP_INT_MAX	获取PHP中整型数的最大值2147483647
PHP_INT_SIZE	获取PHP中整型数的字长，如：4
E_ERROR	表示运行时致命性错误，使用1表示
E_WARNING	表示运行时警告错误（非致命），使用2表示
E_PARSE	表示编译时解析错误，使用4表示
E_NOTICE	表示运行时提醒信息，使用8表示

魔术常量：以双下划线**开始，也以双下划线**结束

```
__DIR__:当前运行文件的所在绝对路径（directory）
__FILE__:当前运行文件的绝对路径（带文件名）
__LINE__:当前代码所在的行号
__FUNCTION__:函数内部使用，表示当前函数的名字
__CLASS__:类内部方法里使用，表示当前类的名字
__METHOD__:类内部方法里使用，表示当前方法的名字（带类名）
__NAMESPACE__:当前所属命名空间的名字
```

示例：

```
#固定常量
echo PHP_VERSION;
echo '<br/>';
echo PHP_INT_SIZE;
echo '<br/>';
echo PHP_INT_MAX;
echo '<br/>';
#魔术常量
echo __DIR__,'<br/>';
echo __FILE__,'<br/>';
echo __LINE__,'<br/>';
```

小结

1、系统内部为我们定义了很多常量

- 固定常量：值是不变的
- 魔术常量：值是不确定的（性质确定）

2、魔术常量在开发是很有用

2.3 数据类型

提问：PHP变量赋值的时候有的地方有引号，有的不需要引号，为什么呢?

回答：数据类型

目标：了解PHP中数据分类的依据和对应的类型

概念

数据类型：数据类型在数据结构中定义是一组 **性质相同的值的集合**以及定义在这个值集合上的一组操作的总称。

- PHP变量没有数据类型：弱类型，可以存储任何数据
- 只有数据是进行分类的

PHP中数据类型分为三大类八小类:

- 基本数据类型
- [ ] 整型（int/integer）:整数数据
- [ ] 浮点型（float）:小数数据和超过整型范围的整型数据
- [ ] 布尔型（bool/boolean）：布尔类型数据，只有true和false两个值
- [ ] 字符串型（string）：字符串数据（最常见）
- 复合数据类型
- [ ] 数组型（array）：数组数据（即多个数据存放到一个变量中）
- [ ] 对象型（object）：对象数据,由类实例化得到（面向对象）
- 特殊数据类型:
- [ ] 资源型（resource）：资源数据（数据库资源\文件资源）
- [ ] 空型（null）：没有存储任何数据

示例

利用var_dump()可以查看变量的详细情况，数据大小（或长度），数据类型

```
<?php
$a=100;
$b='hello';
var_dump($a,$b);
?>
```

输出结果:

```
int(100) string(5) "hello"
```

#int(100) # int类型，值为100

#string类型，长度5个字节，值为hello

小结

1、数据分了是根据数据的性质的同一性进行分类

2、PHP中将数据分成8类:

- 整型：整数
- 浮点型：小数和超出整数范围的整数
- 布尔型：是和否
- 字符串型：字符串数据
- 数组型：数组数据
- 对象型：对象数据
- 资源型：资源数据
- 空型：没有数据

3、可以使用var_dump来查看数据的具体类型和结果详情

4、对象型、资源型和空型需要学习到后面知识，暂时不细讲。

2.3.1 数据类型的分类

1、布尔类型

目标：了解布尔类型的值，掌握布尔类型的应用意义

提问：程序中如何表达行与不行呢?

回答：布尔值

概念

布尔类型：专门用来做判定结果的，只有两个值

true：表示真（是）

false：表示假

示例

```
#布尔表达式

$res=true;
$res=FALSE;
```

注意：布尔值通常不会用来做结果保存，而是用来进行过程判定

小结

1、布尔类型只有两个值：true和false

2、布尔值不区分大小写：true和TRUE一样

3、布尔类型通常不是用来做存储数据，而是程序用来进行过程或者结果判定使用

2、整型

概念

整型：数据是整数，但是根据进制方式不同，不同的数据格式

- 十进制：默认，逢十进一，有0-9十个符号，阿拉伯数字即可，如123
- 八进制：逢八进一，有0-7共八个符号，起始使用0标志，如0123
- 十六进制：逢十六进一，有0-9、a-f共16个符号，起始使用0x标志，如0x123
- 二进制：逢二进一，有0-1共两个符号，起始使用0b标志，如0b101

```
#整型
<?php
$int1=123;
$int2=0123;
$int3=0x123;
$int4=0b101;
echo $int1,'<br/>',$int2,'<br/>',$int3,'<br/>',$int4;
```

123
83
291
5

虽然定义数据的时候可以使用多种进制表示，但是显示数据的时候，默认PHP都会转换成10进制输出

如果想要保持原来的进制输出，就需要使用printf（格式，数据1，数据2...）来实现，例：

```
<?php
$int1=123;
$int2=0123;
$int3=0x123;
$int4=0b101;
echo $int1,'<br/>',$int2,'<br/>',$int3,'<br/>',$int4,'<br/>';
printf('%d',$int1);
printf('%o',$int2);
printf('十六进制是%x,二进制是%b',$int3,$int4);
```

注意：整数最大保存空间为PHP_INT_SIZE的值，即8个字节（PHP7以前是4个字节），能表示的最大值PHP_INT_MAX的值，最小值PHP_INT_MAX+1的负数（PHP支持正负数）

```
echo PHP_INT_SIZE;      #8
echo PHP_INT_MAX;       #9223372036854775807
```

小结

1、整型就是保存的是整数

2、整数的形式有多种，有不同的定义方式：

- 十进制：直接0-9定义
- 八进制：前面使用0，后跟0-7
- 十六进制：前面使用0x，后面跟着0-9 a-f
- 二进制：前面使用0b，后跟0-1

3、PHP不管定义时采用的是什么整型，输出都默认转换成十进制

4、想要保留原来格式，或者指定格式输出整数，使用printf()

- 十进制：%d
- 八进制：%o
- 十六进制：%x
- 二进制：%b
- 5、PHP中整数是有符号类型（有正负）
- 最大值：PHP_INT_MAX
- 最小值：-PHP_INT_MAX-1

进制转换

目标：了解进制的原理，知道PHP中有进制转换的快捷函数

概念

进制转换：是指根据不同的进制要求，将原始数值（通常指整数）

二进制与十进制互转

二进制与八进制互转

二进制与十六进制互转

八进制与十进制互转

八进制与十六进制互转

十进制与十六进制互转

原理

任意进制转十进制的通用方法

简化

在PHP中，已经考虑到业务的需求，所以PHP提供了一系列函数来实现各类进制转换，以十进制为基准

- bindec：二进制转十进制
- decbin：十进制转二进制
- decoct:十进制转八进制
- octdec：八进制转十进制
- dechex：十进制转十六进制
- hexdec：十六进制转十进制

#示例：十六进制转十进制

hexdec('123') #结果：291

小结

1、进制转换原理可以以10进制为基准互转

2、PHP提供了一套以十进制为基准的转换函数，利用十进制与其他进制组合成单词即可实现函数转换

- 八进制：oct
- 十进制：dec
- 十六进制：hex
- 二进制：bin

3、浮点型

掌握浮点型的概念和注意事项

概念

浮点数:即有小数点的数或者数值超过整型能表达的最大数

小数：1.2或者0.12

科学计数法产生的数：4E3==4*10^3

超过整型的数：PHP_INT_MAX+1

浮点数是有有效位的

精度==>精确度

PHP7中的有效精度是13-14位，即数据如果有效数字超过该值，就会自动被四舍五入

13位：肯定精准

14位：如果只有14位数字，那么精准；如果超出14位，第十四位是四舍五入的结果

PHP7中浮点数最大表示：1.8e308

例：

```php
<?php
$f1=0.12;
$f2=4E3;
$f3=PHP_INT_MAX+1;
echo $f1,'<br/>';
echo $f2,'<br/>';
echo PHP_INT_MAX,'<br/>';
echo $f3;
```

运行结果：

0.12
4000
9223372036854775807
9.2233720368548E+18

浮点数存储原理简介

浮点数分为整数部分和小数部分，转换成二进制的方式有所不同，10.5举例：

整数部分：十进制转二进制正常转即可，如10=>1010

小数部分：小数乘以2，取整数部分，如0.5*2=1.0，取1（如果还可以取，继续乘以2）

浮点数一共用8个字节存储，即64位，其中

- 最左侧第一位是符号位：0表示正数，1表示负数
- 从左侧开始第2位-54位存储数值
- 剩余9位存储幂：即10的N次方，N最大为308

扩展

1、浮点数适合运用在精度要求不高的地方（如果确定精度不会超过14位，那么浮点数是精准的）

2、浮点数不适合进行运算后的精确比较（因为小数部分转换成二进制的时候是*2的出来的，取不尽就丢失精度）

- 3与8.1/2.7比较，计算机是不会算出相等的结果的

小结

1、浮点数是用来表示小数，科学计数法数以及超过整数大小的整数的

2、浮点数是有精度范围的，PHP7中精度为14位有效数字

3、浮点数采用8个字节存储，存储原理为：符号位+54位数值位+9位指数位

4、浮点数的应用不建议在精度要求较高（超过14位）的地方使用

5、浮点数不适合进行精确比较（浮点数运算结果）

4、字符串型**

目标：掌握字符串的不同定义方式，以及各个方式的区别

概念

字符串：字符串是由数字、字母、下划线组成的一串字符

字符串是编程中用到最多的数据格式，有多种定义方式

单引号包裹：使用英文单引号 ''包裹字符

双引号包裹：使用英文双引号""包裹字符

定界符包裹：使用PHP中的定界符<<<标识符+标识符结尾形式包裹（多行字符串的一种形式）

heredoc结构：双引号的一种多行字符串形式

nowdoc结构：单引号的一种多行字符串形式

示例

```
#单引号字符串
$str1='hello world';
#双引号字符串
$str2="hello world";
#定界符：heredoc：注意EOT并非固定，自己取名即可（但不要与关键字同名）
$str3= <<<EOT
    $str1
EOT;
echo $str3;
#输出：hello world

#定界符：nowdoc
$str4=<<<'EOT'
    $str1
EOT;
echo $str4;
#输出：$str1
```

区别：

1、单引号是最简单字符串

- 里面能解析少量转义符：单引号（'），反斜杠（\）

```php
#单引号解析
$a=100;
$str='abcdefg\r\n\t\'\\"\\$a';
echo $str;
```

结果：

abcdef\r\n\t'\"\$a

2、双引号相对单引号功能较多

- 里面能解析较多转义符：$符号（$）,双引号（"）,反斜杠（\）,回车换行符（\r\n）,tap符（\t）
- 里面如果有PHP变量，也可以被解析

```php
<?php
$a=100;
$str="abcdef\r\n\t\'\"\\$a";
echo $str;
```

结果：

abcdef \'"\100

```
#\r\n转换成换行空格，单引号不解析\',双引号解析",反斜杠解析\，$a被解析为100
```

3、定界符中heredoc与双引号效果一致，nowdoc与单引号一致

定界符会自动保留格式：即如果内部字符串有换行之类的会自动保留

注意

1、双引号解析变量时，系统是有规矩匹配变量的

- 变量名不要与其他字母或者数字保留在一起：如“$abc"系统会自动识别abc，而不是a
- [ ] 解决方案1:让变量与后面的内容之间有空格，如abc“系统识别a（此时会额外多出一个空格）
- [ ] 解决方案2：使用{}将变量独立包裹，如“{$a}bc”系统识别$a(建议)

2、使用定界符定义多行字符串时有很多规矩

- 定界符内的所有内容都是字符串的内容（包括注释）

```php
$str=<<<END
    #这里是字符串内部
    'abc'
END;
echo $str;
```

- 定界符的起始符号之后不能有任何信息，包括空格

```
$str=<<<END    #注释
    #这里是字符串内部
    'abc';
END;
echo $str;
```

- 定界符的结束符号必须顶格

```
$str=<<<END    #注释
    #这里是字符串内部
    END;
echo $str;
```

- 定界符的结束符号后同样不能有任何信息

```
$str=<<<END    #注释
    #这里是字符串内部
    END;
echo $str;
```

小结

1、字符串是程序编码中应用最多的一种类型，可以使用多种方式定义

- 单引号定义
- 双引号定义
- 定界符定义
  - heredoc定义：代替双引号
  - nowdoc定义：代替单引号

2、单引号与双引号有区别：

- 单引号解析的转义符号较少：'和\
- 双引号解析的转义符号较多：",\、\r\n、\t、$
- 双引号中可以解析变量，但要注意变量的规则
- 让变量名与其他可能让系统误解的符号分开
- 给变量名增加一个{}

3、使用定界符的时候要注意定界符的严格要求

- 起始符号后不能有任何内容（包括空格）
- 结束符号必须顶格
- 结束符号后不能有任何内容

2.3.3 类型判定

目标：掌握类型判定的意义，了解类型判定的常用方法

提问：PHP在进行数据操作的时候，需要用到某些特定类型的数据，PHP如何知道呢?

回答：类型判定

概念

类型判定：即对存储数据的变量（本质是数据）进行类型确定

PHP提供了一套类型判定的函数，以供使用

函数不需要拼命记住，都是以is+数据类型组成

函数在操作手册中都有，输入is_即可插到所有

常用类型判定

系统提供的类型判定很全，有几个特殊的需要了解一下

- is_numeric:判定数据是否是数值，如果是返回布尔TRUE，否则返回FALSE
- is_scalar:是否是标量（基础类型）数据，如果是返回布尔TRUE，否则返回FALSE
- 类型代表：is_int,判定是否是整型数据，如果是返回布尔TRUE，否则返回FALSE

```php
<?php
$num=100;
$str='100';
var_dump($num,$str);
var_dump(is_int($num),is_string($str));
var_dump(is_numeric($num),is_numeric($str));
var_dump(is_scalar($num),is_scalar($str));
```

小结

1、类型判定是因为某些特定情况下，需要保证数据格式能对上

不要相信任何外来数据，保证程序的安全和逻辑完整性

类型判定是用到PHP提供的系统函数，函数以is_+数据类型

- is_int:判定整型（is_integer）
- is_float:判定浮点型
- is_bool:判定布尔类型（is_boolean）
- is_string:判定字符串型
- is_null:判定空型
- is_array:判定数组类型
- is_resource:判定资源型
- is_object:判定对象型

3、常见的数据类型判定，有两个额外的判定函数

is_numeric:判定是否是数组或者数值型字符串

is_scalar:判定是否是基础数据类型（整型、浮点型、布尔型、字符串型）

2.3.4 类型转换

目标：了解类型转换的原因，掌握类型转换的逻辑

提问：如果某项数据的类型如字符串100，而要参与计算的是数值运算，要怎么办?

回答：类型转换

概念

类型转换：即将某种不符合的类型转换成目标类型

自动转换：系统自动转成目标类型（方便）

步骤

1、自动转换

- 系统自动检测所需要的类型

- 系统自动检测数据的类型
- 系统自动转换成目标类型（不改变数据本身）

2、强制转换

- 明确所需要类型
- 强制转换成所需类型
- [ ] 不改变原来数据：在数据前使用（目标类型），如（bool）a，将变量a的值取出来转成布尔结果
- [ ] 改变原来数据：使用settype（变量，'目标类型'），如settype(a,'bool'),将变量a的值转成布尔结果
- 强制转换的类型：资源和NULL不能强制转换
- [ ] integer：转换成整型
- [ ] float：转换成浮点型
- [ ] bool：转换成布尔类型
- [ ] string：转换成字符串类型
- [ ] array：转换成数组类型
- [ ] object：转换成对象类型

```php
<?php
$num=4;
$str='1.23a';
#自动转换
echo $num+$str;
var_dump($str);
#强制类型转换
echo (float)$num+(float)$str;
#真正转换类型
$res=settype($str,'float');
var_dump($res,$str);

?>
```

常见的转换逻辑

1、字符串转数值

- 纯数值字符串=》相应数值：'123.1'=>123.1
- 数字开头担忧字母=》保留数字部分：'12a'=》12
- 小数点开头=》保留第一个小数点及之后的连续数字：'.1.1.a'=>0.1
- 字母开头=》0：'a123'=>0

2、其他类型转布尔：极少转换后是FALSE，基本都是TRUE，以下除外

- 布尔FALSE转布尔：FALSE
- 整型0转布尔：FALSE
- 浮点型0.0转布尔：FALSE
- 空白字符串和字符串'0'转布尔：FALSE
- 空数组array()转布尔：FALSE（比较多运用）
- 空类型NULL转布尔：FALSE

2.4 运算符

目标：了解PHP中运算符的分类

提问：PHP中拿到了各种数据之后，是用来干什么的?

回答：运算

概念

运算符：是PHP中的一些特殊符号，系统会自动根据这些特殊符号，将数据进行相应的运算

PHP中运算符有很多种：

- 赋值运算符：赋值操作
- 算术运算符：算术计算
- **错误抑制符：抑制报错**
- 比较运算符：数据对比
- 合并运算符：振幅判定给出不同结果
- 逻辑运算符：逻辑结果判定
- 连接运算符：字符串连接
- 三目运算符：真伪判定给出不同结果
- 自操作运算符：简化操作
- 位运算符：根据计算机最小的单位bit进行运算

步骤

1、确定当前需要使用到的运算符号

2、根据运算符号的规则确定需要参与的表达式个数

3、将对应的表达式和符号按照正确的顺序排列即可

2.4.1算术运算符

算术运算符：用来进行数学运算的符号。

运算符	作用	范例	结果
+	加	echo 5 + 5;	10
-	减	echo 6 - 4;	2
*	乘	echo 3 * 4;	12
/	除	echo 5 / 5;	1
%	取模（即算术中的求余数）	echo 7 % 5;	2

注意：运算顺序要遵循数学中“先乘除、后加减”的原则；

　取模运算时，运算结果的正负取决于被模数（%左边的数）的符号，与模数（%右边的数）的符号无关。例如：（-8）%7=-1，而8%（-7）=1。

补充：**：幂运算，PHP后新增，底数的指数次相乘。

2.4.2赋值运算

赋值运算符：它是二元运算符，有两个操作数，用来把赋值运算符右边的值赋给左边的变量。

运算符	作用	范例	结果
=	赋值	$a = 3; $b = 2;	$a = 3; $b = 2;
+=	加并赋值	$a = 3; $b = 2; $a += $b;	$a = 5; $b = 2;
-=	减并赋值	$a = 3; $b = 2; $a -= $b;	$a = 1; $b = 2;
*=	乘并赋值	$a = 3; $b = 2; $a *= $b;	$a = 6; $b = 2;
/=	除并赋值	$a = 3; $b = 2; $a /= $b;	$a = 1.5; $b = 2;
%=	模并赋值	$a = 3; $b = 2; $a %= $b;	$a = 1; $b = 2;
.=	连接并赋值	$a = 'abc'; $a .= 'def';	$a = 'abcdef';

```php
<?php
#普通赋值
$a=3;
$b=2;
$c='string';
#复合赋值
$a+=$b;
$c.='hello';
echo '<pre>';
var_dump($a,$c);
```

注意

- 复合赋值运算是将左侧与右侧的整体结果进行运算

```
$a=10;
$a-=100-90;
echo $a; #结果为0
```

当复合赋值是/=或者%=的时候，右侧的结果不能为0

```
$a=10;
$a/=0;
```

2.4.3 【案例】商品价格计算

若用户在一个全场8折的网站中购买了2斤香蕉、1斤苹果和3斤橘子，它们的价格分别为7.99元/斤、6.89元/斤、3.99元/斤，如何使用PHP程序来计算此用户实际需支付的费用呢?

请通过PHP中提供的变量、常量、算术运算符以及赋值运算符等相关知识实现PHP中的商品价格计算。

作业1：完成案例商品价格计算代码的编写，并将运行结果上传到对分易。

2.4.4错误抑制

目标：了解错误抑制符的作用，灵活运用错误抑制符来实现可能出现的错误的抑制

提问：当进行算术运算的时候如果除数为0就错了，该怎么控制呢?

回答：数据判断或者暴力错误抑制

概念

错误抑制：在可能出现的错误的代码前使用错误抑制符，让错误不会报出来

错误抑制只需要在可能出现的错误的表达式之前使用即可

错误抑制只有一个：@

错误抑制符只针对就近的目标，如果是针对结果，需要使用（）

示例：

```
$a=@(10/0);
```

小结

1、错误抑制符是用来抑制可能出现的错误的

抑制级别：notice、warning低级别错误

2、错误抑制符@应该用在出现错误的位置

- 如果只是一个表达式错误：那么直接@
- 如果错误是一个过程：那么需要先将过程使用（）包起来，然后进行抑制

2.4.5比较运算

目标：了解比较运算的意义和工作原理，掌握常用比较运算符的使用

提问：如何知道两个数据的大小呢?

回答：比较运算

运算符	运算	范例	结果
==	等于	5 == 4	false
!=	不等于	5 != 4	true
<>	不等于	5 <> 4	true
===	全等	5 === 5	true
!==	不全等	5 !== '5'	true
>	大于	5 > 5	false
>=	大于或等于	5 >= 5	true
<	小于	5 < 5	false
<=	小于或等于	5 <= 5	true

示例

```php
<?php
$a=10;
$b=5;
$c='10';
echo '<pre>';
var_dump($a>$b);
var_dump($a==$b);
var_dump($a==$c);
var_dump($a===$c);
```

注意：在PHP中比较运算比较特殊，通常系统会自动将类型转换成相同的然后进行比较。

"==="与"!=="运算符在进行比较时，不仅要比较数值是否相等，还要比较数据类型是否相同。"=="和"!="运算符在比较时，只比较数值是否相等。

2.4.6合并运算符

合并运算符：PHP 7新增的运算符，用于简单的数据存在性判定。

语法格式

<条件表达式> ?? <表达式>

代码示例

```php
$age = $age ?? 18;
```

如果$age存在，则使用$age的值，如果$age不存在，则将$age的值设置为18。

2.4.7三元运算符

三元运算符：又称为三目运算符，它是一种特殊的运算符。

<条件表达式> ? <表达式1> : <表达式2>

```php
$age = 10;
echo $age >= 18 ? '已成年' : '未成年';
```

如果变量$age的值大于或等于18，输出结果为“已成年”，如果小于18，则输出结果为未成年。

2.4.8逻辑运算符

逻辑运算符：用于逻辑判断的符号，其返回值类型是布尔类型。

运算符	运算	范例	结果
&&	与	$a && $b	$a和$b都为true，结果为true，否则为false
||	或	$a || $b	$a和$b中至少有一个为true，结果为true，否则为false
!	非	!$a	若$a为false，结果为true，否则相反
xor	异或	$a xor $b	$a和$b一个为true，一个为false，结果true，否则false
and	与	$a and $b	与“&&”相同，但优先级较低
or	或	$a or $b	与“||”相同，但优先级较低

```php
<?php
#逻辑运算
$age=21;

#判定年龄
$res=$age<18 ||$age>65;
var_dump($res,!$res);

#短路验证
$age>65 || $b=10;
var_dump($b);

#$a=false or die('错误');
echo $a;
```

2.4.9递增/递减运算符

递增递减运算符：也称为自增自减运算符，是一种特定形式的复合赋值运算符。

运算符	运算	范例	结果
++	自增（前）	$a = 2; $b = ++$a;	$a = 3; $b = 3;
++	自增（后）	$a = 2; $b = $a++;	$a = 3; $b = 2;
--	自减（前）	$a = 2; $b = --$a;	$a = 1; $b = 1;
--	自减（后）	$a = 2; $b = $a--;	$a = 1; $b = 2;

示例：

```php
<?php
$a=$b=1;
$a++;
++$b;
echo $a,$b;   #独立运算没有前后之分
$a=$b=0;
echo '我今年：'.$a++.'岁<br/>';
echo '我今年：'.++$b.'岁<br/>';
echo $a,$b;
```

小结

1、自操作包含++和--都是一个变量参与的改变运算，每次都改变1

2、自操作运算包含前置和后置运算

前置自操作：先改变自己、后参与运算

后置自操作：先参与运算、后改变自己

条件：前后置的区别需要运算本身还要参与到其他运算

- 输出
- 算术运算

3、自操作运算是有规律的自增1或者自减1，实际在循环时会经常用到（循环条件的变更）。

2.4.10位运算符

位运算符：针对二进制数的每一位进行运算。

运算符	名称	范例	结果
&	按位与	$a & $b	$a和$b各二进制位进行“与”操作后的结果
|	按位或	$a | $b	$a和$b各二进制位进行“或”操作后的结果
~	按位非	~$a	$a的各二进制位进行“非”操作后的结果
^	按位异或	$a ^ $b	$a和$b各二进制位进行“异或”操作后的结果
<<	左移	$a << $b	将$a各二进制位左移b位（左移一位相当于该数乘以2）
>>	右移	$a >> $b	将$a各二进制位右移b位（右移一位相当于该数除以2）

位运算符可以对整型和字符串进行位运算：

在对数字进行位运算之前，程序会将所有的操作数转换成二进制数，然后再逐位运算。

在对字符进行位运算之前，首先将字符转换成对应的ASCII码（数字），然后对产生的数字进行运算，再把运算结果（数字）转换成对应的字符。

```
<?php
$a=5;
$b=3;
echo $a &$b;
echo $a |$b;
echo $a ^$b;
echo ~$a;
```

1、PHP中的整数是用8个字节存储，一共64位。

2、计算机码分为原码、反码、补码。

3、正数的原码、反码、补码就是原码不变，负数是以补码的形式存储。

4、负数最终显示的结果就是将补码-1，变成反码，然后取反（符号位不变），变成原码。

5、系统存储的是补码，用户看到的结果都是原码转换的结果。

```
$a=5
#按位取反
echo ~a;

00000000.....00000101
11111111.....11111010  反码
11111111.....11111001  补码:反码-1
10000000.....00000110  原码:符号位不变，其他位取反，最终结果为-6
```

2.4.11运算符的优先级

运算符优先级：一个表达式中有多个运算符，这些运算符会遵循一定的先后顺序。

结合方向	运算符	结合方向	运算符
无	new	左	^
左	[	左	|
右	++ -- ~ (int) (float) (string) (array) (object) @	左	&&
无	instanceof	左	||
右	!	左	? :
左	* / %	右	= += -= *= /= .= %= &= |= ^= <<= >>=
左	+ - .	左	and
左	<< >>	左	xor
无	== != === !== <>	左	or
左	&	左	,

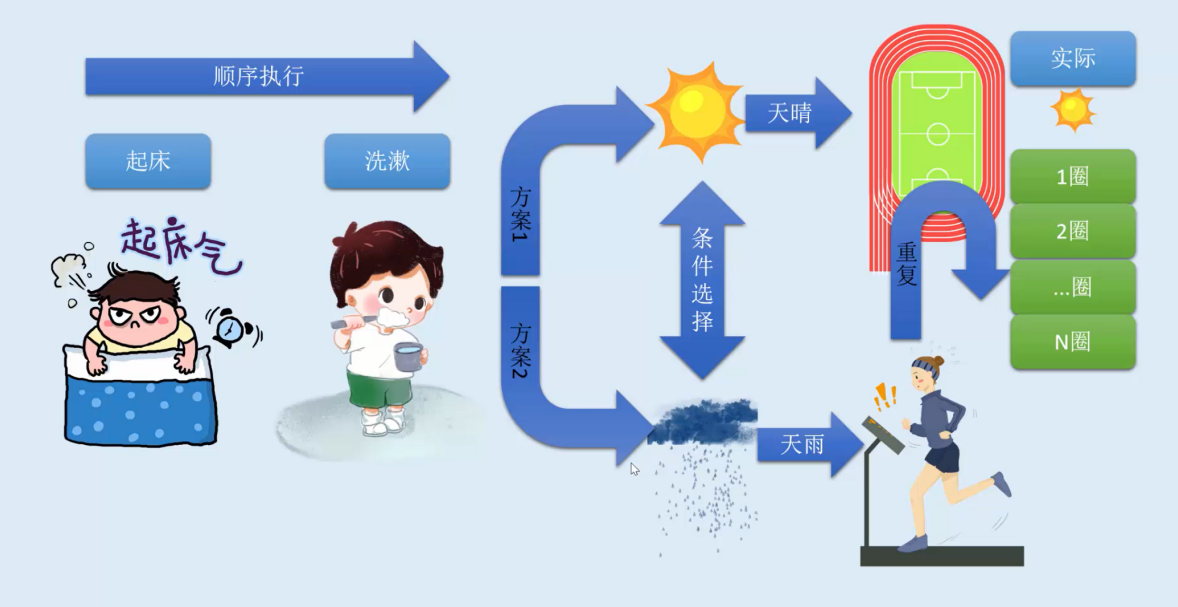
改变运算符的优先级：在表达式中使用小括号“( )”可以提升运算符的优先级。

改变运算符优先级代码示例

```
$num1 = 4 + 3 * 2;      // 输出结果为10
$num2 = (4 + 3) * 2;    // 输出结果为14
```

2.5流程控制

举例子



说明：我们每天都在做一些类似的事情

1、顺序执行某些事情：起床后刷牙

2、在规划后选择某一件执行：两个晨练方案，根据天气条件选择执行

3、在做某一件事情之后重复执行：跑圈圈

概念

流程控制：就是设定好代码的执行流程，流程控制有三大结构

顺序结构：代码逐行向下顺序执行

分支结构：设定好代码，根据条件选择性执行

循环结构：设定好代码，根据条件重复执行

小结

1、流程控制是代码为了实现现实的业务场景而设计的代码运行方式

2、流程控制分为三种

顺序结构：代码顺序执行

分支结构：代码选择性执行

循环结构：代码重复执行

1、顺序结构

目标：了解代码的顺序执行方式

概念

顺序结构：即代码是以行为单位，逐行向下执行的

示例

```
$a=10;
echo $a;
```

以上执行逻辑

1、执行第一行：$a=10,在内存产生变量

2、执行第二行：echo $a, 输出10（空行忽略）

3、执行到最后：脚本执行结束，销毁所有资源

小结

1、代码默认是顺序结构执行，即逐行向下

2、分支结构

目标：掌握分支结构的基本逻辑，掌握分支结构的实际运用

概念

分支结构：即在编写代码的时候，考虑到各种情形，将不同情形的代码实现分块。在实际运行的时候根据条件选择相应的代码块执行（一般职执行一块代码）

- IF分支：典型的互斥分支，根据条件选择一块代码执行

- Switch分支：粒子分支，条件的颗粒度较强，可以选择一块或者多块执行

分支结构是编程中非常常见的代码结构

步骤

1、确定条件（通常可变）

2、设置不同条件的代码

3、根据条件执行代码

小结

1、分支结构是要提前准备条件和多个代码块（有可能只有一个代码块）

2、分支执行是根据条件选择代码块执行

2.1、IF分支

目标：掌握IF分支的结构语法，以及能够运用IF结构实现分支结构解决问题

概念

IF分支：一种可以接收任意条件的分支结构

- [ ] 简单IF分支：条件满足执行，不满足不执行代码
- if(条件表达式){满足条件后执行的代码块}
- [ ] 标准IF分支：条件满足执行一块代码，不满足执行另外一块代码
- if(条件表达式){满足条件后执行的代码块}
- else{不满足条件后执行的代码块}
- [ ] 复杂IF分支：有多个条件，条件有递进性
- if(条件1){满足执行}
- elseif（条件2）{不满足条件1满足条件2执行} ..
- else{都不满足执行}
- [ ] IF分支永远最多执行一个代码块
- [ ] IF分支代码块李可以嵌入IF分支

步骤

1、确定条件

条件是否唯一

2、确定分支代码块

满足条件的代码块

不满足条件的代码块（非必须：看功能需要）

3、根据条件执行

示例

1、明天天气好，我就出去走走

```
#确定条件
$weather='good';

#确定分支代码块
if($weather=='good'){
    echo '出去走走';
}

#结果：出去走走
#执行时：如果$weather=='good',执行代码块，否则跳过
```

2、明天天气好，我就出去走走；否则我就在家睡觉

```
#确定条件
$weather='rain';

#确定分支代码块
if($weather=='good'){
    echo '出去走走<br/>';
}else{
    echo '在家睡觉<br/>';
}
```

3、明天天气出太阳，出去走走；天气刮风，去看电影；天气阴天，在家看书；否则睡觉

```
#确定条件
$weather='wind';

#确定分支代码块
if($weather=='sun'){
    echo '出去走走<br/>';
}elseif($weather=='wind'){
    echo '去看电影<br/>';
}elseif($weather=='cloud'){
    echo '在家看书<br/>';
}else{
    echo '睡觉<br/>';
}
```

4、明天天气好：出太阳，出去走走；其他，去看电影；天气不好：阴天，看电视；其他，睡觉

```
#确定条件：两组条件（大条件和小条件）
$weather1='good';
$weather2='sun';
#确定分支代码块：大条件
if($weather1 =='good'){
    if($weather2='sun'){
        echo '出去走走<br/>';
    }else{
        echo '去看电影<br/>';
    }
```

```
}else{
    #天气不好：确定小条件
    if($weather2=='cloud'){
        echo '看电视<br/>';
    }else{
        echo '睡觉<br/>';
    }
}
```

2.2、Switch分支

目标：掌握switch分支的结构语法，以及能够运用switch结构实现分支结构解决问题

概念

Switch分支：一种接收颗粒条件的分支结构

switch分支：条件是具体值，而不是范围条件

Switch代码块是写在一起的，使用case分开，而不是{}

Switch代码块允许执行多个（如果有必要）

Switch结构

switch（条件变量）{

case 值：

满足条件1要执行的代码块

代码执行控制：break

……

default:

不满足所有条件执行的代码块

代码执行控制：break

}

步骤

1、确定目标条件：条件必须是颗粒度的，通常是一个变量（没有比较符号）

2、确定可能出现的结果：一个条件对应一个case指令

3、确定每个case指令下要执行的代码

4、确定每个case指令是否需要结束

结束使用break

不结束，代码会继续执行下一个case的代码块（代码块共享）

5、确定是否需要不满足条件的代码块：default

示例

1、明天天气出太阳，出去走走；天气刮风，去看电影；天气阴天，在家看书；否则睡觉

```
```

```
#假设外部接收条件：条件变量为$weather,值为sun、wind、cloud和其他
switch($weather){
    #匹配第一个值：直接取$weather的值即可
    case 'sun':
        echo '出去走走';
        break;
    case 'wind':
        echo '去看电影';
        break;
    case 'cloud':
        echo '在家看书';
        break;
    default:
        echo '睡觉';
        break;
}
```

2、一周7天，1-5天上班，6和7休息（共享代码块）

```
#switch分支结构
#一周7天，一周7天，1-5天上班，6和7休息
$day=2;
switch($day){
    case 1:
        echo '周一，上班';
        break;
    case 2:
    case 3:
    case 4:
    case 5:
         echo '上班';
         break;
    case 6:
         echo '看电影';
         break;
    case 7:
         echo '旅游';
         break;
    default:
        echo '太好了';
        break;
}
```

小结：

1、switch是一种用来实现颗粒条件的分支结构

只适合固定值：case 具体值

不适合进行大小比较：不能case条件>10之类

2、switch的结构如下

```
switch(条件变量){
    case 值1:
        代码块;
        break;
    case 值2:
        代码块;
        break;
    default:
        代码块;
        break;
}
```

3、break如果不写，那么可以实现多个case共享一个代码块

4、default可以没有

5、凡是switch能做的分支，if都能做；但是if能做的，switch不一定能做

3、循环结构

目标：掌握循环的结构语法，掌握循环的实际运用

概念

循环结构：指通过某种条件的限定，然后某个代码块进行可控的重复的执行

循环结构一般包含以下几个部分

- 循环条件：判断循环是否执行
- 循环变量变化：让条件持续改变
- 循环体：要重复执行的代码块
- 循环控制：内部对循环的控制

PHP中循环结构有以下几个：

while循环：不确定条件的重复执行

for循环：一般指定次数的重复执行

foreach循环：专门遍历数组

do-while循环：了解

循环控制：内部控制循环的执行

步骤：

1、根据循环条件确定要重复的内容，选择合适的循环结构

2、确定循环条件的边界

3、确定循环条件的变更方式

4、确定循环体（要重复执行的代码块）

5、确定循环内部控制（不一定要）

小结

1、循环结构是一种让代码在可控范围内执行N次的流程结构

2、循环结构包含多种

while循环：不确定条件的重复执行

for循环：一般指定次数的重复执行

foreach循环：专门遍历数组

do-while循环：了解

3、循环也可以在内部进行循环控制

3.1、while循环

目标：掌握while循环的基本语法，熟练运用我while循环实现内容重复执行

提问：想输出1-10可不可以不是10次echo呢?

回答：可以，1到10是有规矩，找一个东西从1变到10 就好

概念

while循环：在指定边界条件下，持续执行代码的一种循环

- while循环需要边界条件（也可以没有）
- while条件需要在内部控制循环变量
- while循环多用来实现一些条件不大明确的循环

语法结构：

```
while(表达式){
    循环体;
    循环条件变更;
}
```

步骤

1、通常会在循环开始前定义一个循环变量的初始值

2、确定循环条件的边界条件

3、确认循环体（循环要干嘛）

4、变更循环条件

示例

```php
#输出1-10

#1、确定循环变量初始值：可以是1，也可以是10
$i=1;

#2、确定循环边界条件：10
while($i<10){
    #3、确定循环体：输出1-10
    echo $i;

    #4、变更循环条件
    $i++;
}
```

3.2 、for循环

目标：掌握for循环的语法结构，熟练使用for循环实现代码重复执行

for循环：一种将循环初始条件、循环边界条件和循环条件变更都可以放到循环结构里（不是循环体）的循环

- for循环需要循环初始条件，但可以在循环结构里
- for循环需要边界条件，也可以放在循环结构里
- for循环进行循环条件变更，也可以放在循环结构里
- for循环一般用来实现一些固定起始边间和步长（条件变化的规律）的循环，多与数字有关

for语法结构：

```php
for(初始化条件表达式；边界判定表达式；条件变更表达式){
    循环体
}
```

示例

1、输出1-100之间的所有奇数

```php
for($i=1;$i<=100;$i++){
    if($i%2==1){
        echo $i;
    }
}
```

2、循环嵌套：打印九九乘法表

```php
#1、确定一个循环，能够打印9行
for($i=1;$i<=9;$i++){
    #2、确定一个循环：能够输出9列
    for($j=1;$j<=i;$j++){
        echo "$j*$i=".$j*$i.'  ';
    }
    echo '<br/>';
}
```

3.3、 do-while循环

目标：了解do-while的基本语法以及执行原理，了解do-while与while的区别

概念

do-while循环：是一种先执行循环体（do）然后再考虑条件的循环

do-while 循环需要边界条件

do-while循环在内部变更循环条件

do-while用来处理不明确条件的循环

do-while语法

do{

　循环体

　循环条件变更

}while（循环条件判定）；

1、通常在循环开始前定义一个循环变量的初始值

2、确认循环体（循环要干嘛）

3、变更循环条件

4、确定循环条件的边界条件

示例

输出1-10

```
$i=1;
do{
    echo $i,'<br/>';
    $i++
}while(i<=10);
```

do-while与while循环的区别

- while是先判定条件后执行，do-while是先执行后判定条件
- do-while一定会保证循环体知识循环一次，而while未必会执行循环体。

在PHP中do-while基本不使用，了解即可。

3.4、 跳转语句

跳转语句：用于实现循环执行过程中程序流程的跳转。

常用的跳转语句：break语句和continue语句。

break用于终止当前循环，跳出循环体。

continue用于结束本次循环的执行，开始下一轮循环的执行。

求1-100之间不是3的倍数的整数的和

```php
$i=1;
$sum=0;
while($i<=100){
    if($i%3==0){
        $i++;
        continue;
    }
    $sum+=$i;
    $i++;
}
echo $sum;
```