**常量、流程控制、函数**

目录

[二、 类型自动转换 6](#_Toc495160480)

[1. 自动转换 6](#_Toc495160481)

[1) 转换为布尔值为false的情况 6](#_Toc495160482)

[2) 转换为数值类型的原则： 6](#_Toc495160483)

[2. 类型强制转换 7](#_Toc495160484)

[3. 自动转换及强制转换比较 7](#_Toc495160485)

[三、 常量 8](#_Toc495160486)

[1. 常量的定义 8](#_Toc495160487)

[1) 语法1 8](#_Toc495160488)

[2) 语法2 8](#_Toc495160489)

[2. 常量在内存中的表示形式 9](#_Toc495160490)

[四、 运算符 9](#_Toc495160491)

[1. 什么是运算符 9](#_Toc495160492)

[2. 分类 9](#_Toc495160493)

[1) 按照操作数的数目进行分类 10](#_Toc495160494)

[2) 按照功能： 10](#_Toc495160495)

[3) 运算符优先级 11](#_Toc495160496)

[3. 特殊运算符 11](#_Toc495160497)

[1) 字符串连接符 . 11](#_Toc495160498)

[2) 字符串分隔符 , 12](#_Toc495160499)

[3) 错误抑制符 @ 12](#_Toc495160500)

[4. 算术运算符 12](#_Toc495160501)

[1) 计算中的类型自动转换 12](#_Toc495160502)

[2) 取余运算符% 12](#_Toc495160503)

[5. 赋值运算符= 13](#_Toc495160504)

[1) 功能 13](#_Toc495160505)

[2) 赋值运算符的右结合性 13](#_Toc495160506)

[6. 自操作运算符 13](#_Toc495160507)

[1) ++,--两种使用方法 13](#_Toc495160508)

[7. 比较运算符 14](#_Toc495160509)

[1) 相等、全等比较的本质 14](#_Toc495160510)

[2) 相等比较运算原则 14](#_Toc495160511)

[3) 全等比较 14](#_Toc495160512)

[4) =与==及===的区别 14](#_Toc495160513)

[8. 逻辑运算符 15](#_Toc495160514)

[1) 运算原则 15](#_Toc495160515)

[2) 短路运算 16](#_Toc495160516)

[9. 条件运算符 17](#_Toc495160517)

[五、 流程控制 17](#_Toc495160518)

[1. 程序语言的三大流程控制 17](#_Toc495160519)

[2. 顺序结构 18](#_Toc495160520)

[3. 分支（选择结构） 18](#_Toc495160521)

[1) 条件执行if else 18](#_Toc495160522)

[2) if elseif else 19](#_Toc495160523)

[3) switch语句语法 19](#_Toc495160524)

[4) if与switch区别 21](#_Toc495160525)

[4. 循环结构 21](#_Toc495160526)

[1) while语法 22](#_Toc495160527)

[2) do while循环语法 22](#_Toc495160528)

[3) for循环语法 24](#_Toc495160529)

[5. die,exit 25](#_Toc495160530)

[1) die,exit 25](#_Toc495160531)

[2) sleep休眠函数 25](#_Toc495160532)

[六、 函数语法 25](#_Toc495160533)

[1. 声明语法 25](#_Toc495160534)

[1) 语法 25](#_Toc495160535)

[2) 快速入门：包饺子函数 26](#_Toc495160536)

[3) 五要素 26](#_Toc495160537)

[2. 调用语法 26](#_Toc495160538)

[3. 函数名命名规则 27](#_Toc495160539)

[4. 函数体 28](#_Toc495160540)

[5. 函数的参数 28](#_Toc495160541)

[1) 形参 29](#_Toc495160542)

[2) 默认值 29](#_Toc495160543)

[3) 实参 30](#_Toc495160544)

[6. PHP支持的作用域及特点 30](#_Toc495160545)

[1) 全局作用域 30](#_Toc495160546)

[2) 局部作用域： 30](#_Toc495160547)

[3) 全局作用域与局部作用域的关系 30](#_Toc495160548)

[7. 超全局变量 31](#_Toc495160549)

[七、 系统函数 32](#_Toc495160550)

[1. 时间日期函数 32](#_Toc495160551)

[1) time() 32](#_Toc495160552)

[2) date() 32](#_Toc495160553)

[3) strtotime() 33](#_Toc495160554)

[2. 数学函数 33](#_Toc495160555)

[3. 案例：使用数学函数生成验证码文字 34](#_Toc495160556)

[八、 文件载入 35](#_Toc495160557)

[1. 介绍 35](#_Toc495160558)

[2. 语法 35](#_Toc495160559)

[1) 原理 35](#_Toc495160560)

[2) 引入PHP文件 36](#_Toc495160561)

[3) 引入HTML文件 36](#_Toc495160562)

[3. 路径问题 37](#_Toc495160563)

[1) 相对路径; 37](#_Toc495160564)

[2) 绝对路径 37](#_Toc495160565)

[3) 魔术常量\_\_DIR\_\_ 38](#_Toc495160566)

[4. include与require、include\_once区别 38](#_Toc495160567)

[1) include与require区别： 38](#_Toc495160568)

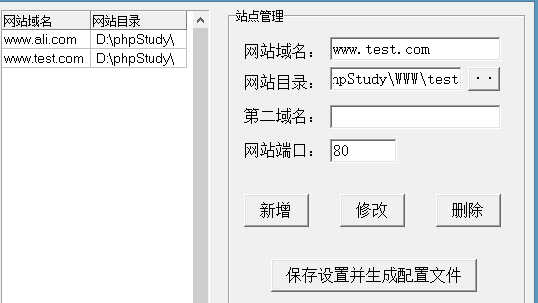
[2) \_once作用： 39](#_Toc495160569)

**回顾：**

1. 创建一个基于域名的虚拟主机，指向www/test文件夹。并设置默认首页为index.php或者为index.html.存在默认首页时，直接展示首页，不存在时以列表的形式返回当前访问目录下的所有文件。

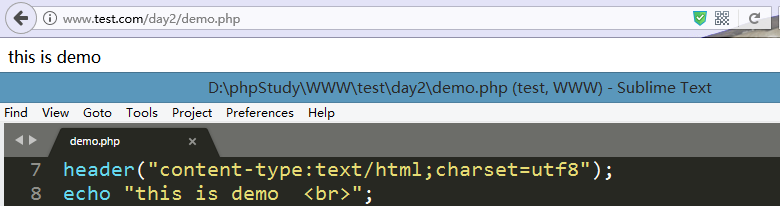


1. 在phpstudy配置

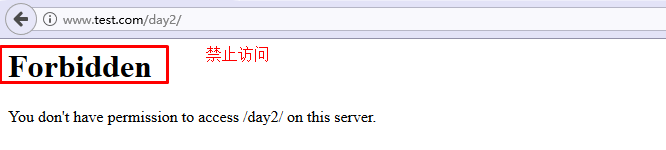


1. 解析[www.test.com](http://www.test.com)

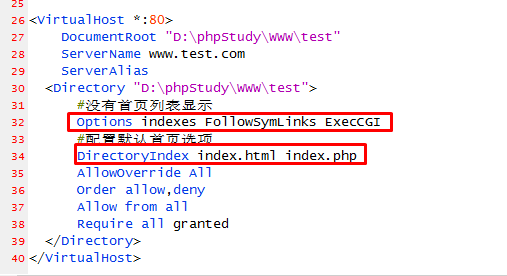


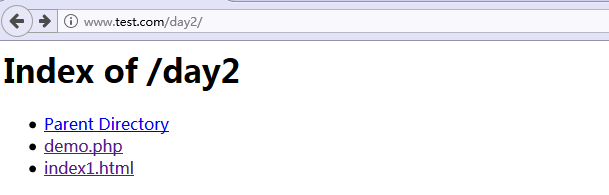


1. 存在默认首页，展示默认首页 ，不存在的，展示所有脚本：



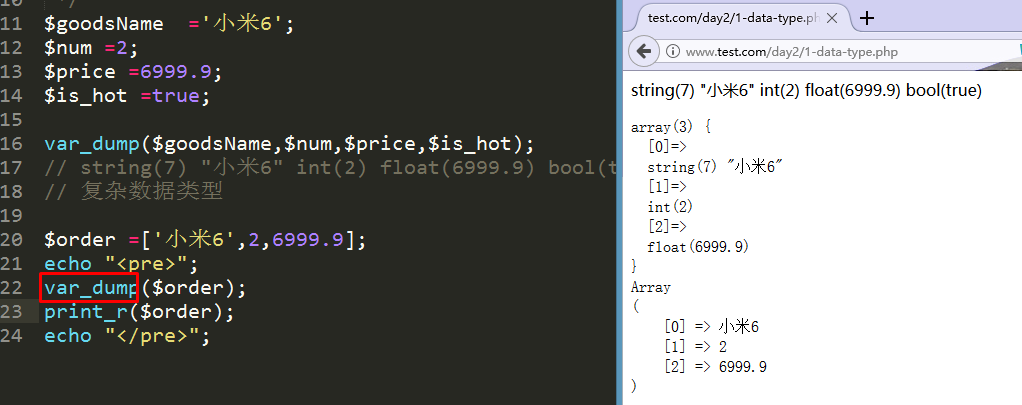
在对应配置段中添加：





1. 创建变量表示PHP的所有数据类型

三大类（标量，复合，特殊），八小类

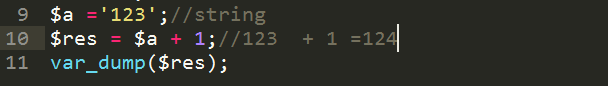


# 类型自动转换

PHP本身为弱类型语言，支持类型的自动转换。也可以通过指定要转换的类型,实现变量的强制转换。

## 自动转换

数据参与运算时，根据实际环境，自动转换参与运算时的数据类型。



### 转换为布尔值为false的情况

在进行条件判断时，由于结果只有满足、不满足，因此一般需要将非布尔类型的表达式的值，转换为布尔类型的true或者false。

**转换后为false的情况：**

* 整型0
* 浮点数0.0
* 字符串’0’
* 空字符串‘’
* 空数组
* null

### 转换为数值类型的原则：

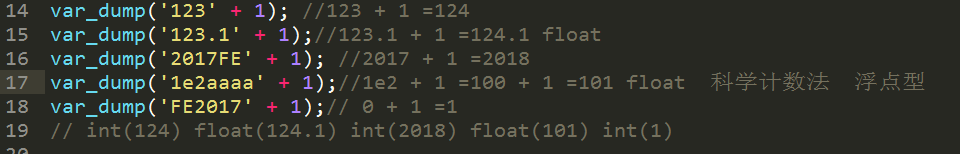
参与算术运算的数据类型一定是数值型。

* 纯数字组成的字符串转换为相应的数字 ‘123’→123
* 可以转换为浮点型的字符，转换为相应的浮点数。 ‘123.1’→123.1
* 以数字开头的字符串，转换为对应的数字

‘2017FE’→2017

‘1e2aaaa’→100（浮点型）

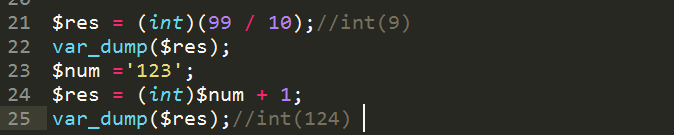
* 不以数字开头的字符串，转换为0.包括字符串’true’。 “FE2017”→ 0
* 布尔值true转换为1，false转换为0.
* 自动转换后不改变原始变量的数据类型，仅修改参与运算的类型



## 类型强制转换

(int | string | bool | float) $var

取整问题：

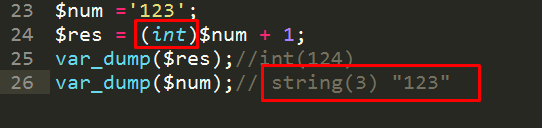


**转换原则：**

* 与类型自动转换原则一致。
* 需要在变量前使用括号指定目标转换类型

## 自动转换及强制转换比较

* 转换之后，原始数据类型不发生转换，只是数据参与运算的值,发生改变



* 自动转换与强制转换的原则一致。

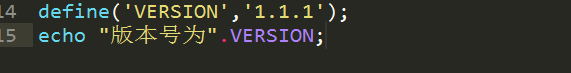
# 常量

## 常量的定义

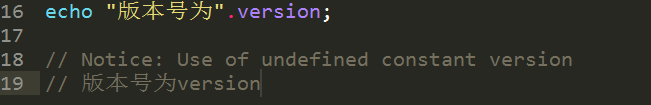
脚本执行周期内，值不会发生改变的量, 英文为：constant。常量不可以修改及删除。相同的值，定义为常量,可以节省存储空间，如运行模式常量，版本常量，路径常量等。

### 语法1

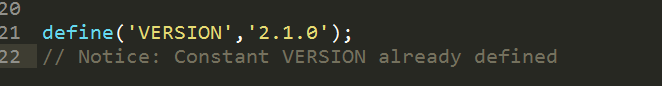
define(常量名，常量值);



* 常量默认区分大小写。



* 按照开发惯例，常量名推荐全部字母大写。
* 常量不可以重复定义及修改数据。

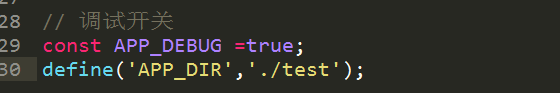


* 常量值为标量数据类型（整型、浮点型、布尔类型、字符串）

### 语法2

const 常量名=常量值

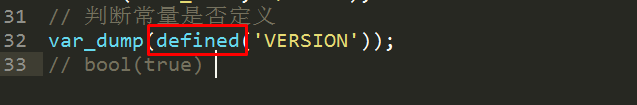
与define定义的常量一样。不能在函数、流程控制（if、while）中定义常量。





## 常量在内存中的表示形式

常量存放在常量区，该区域的量不可以修改。常量一旦定义，不可以重新定义及修改，以保证数据不被修改。常用来存储：版本号，路径等。



defined(‘常量名’);

判断常量是否已经定义，返回true 或者false

# 运算符

## 什么是运算符

运算符连接操作数或者表达式,进行对应的运算。

如

$res = $num1 + $num2 + 算术运算符

var\_dump($num1 == $num2) ==相等比较运算符

## 分类

### 按照操作数的数目进行分类

* 单目运算符：操作数只有1个，变量对自身进行操作。 ++$a, $a--
* 双目运算符：操作数只有2个。+，-，\*, /
* 三目运算符 ：又称条件运算符。

格式：

条件? 表达式1 ：表达式2

即满足条件，返回表达式1的值，否则，返回表达式2的值.

类似于：

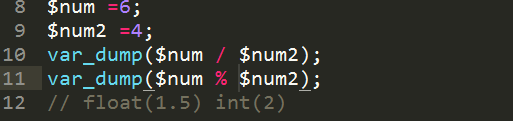
吃了吗 ？ 咖啡 ：面包

### 按照功能：

* + 算术运算符：+, -, \*, /（除法）,%（取模，取余）

6 / 4 =1.5

6 % 4 =2



* + 赋值运算符：=,将表达式右侧的值赋值到左侧
  + 自操作：++，--，+=，-=，\*=，/=,%=

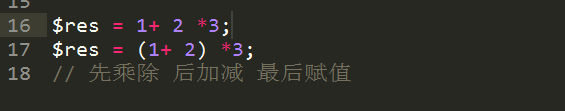
$a =123;

$a++;

$a += 2; //等价于$a = $a + 2;

* + 比较运算符： >,< , ===（全等于）， ==相等。比较的结果是真或者假， 返回true或者false
  + 逻辑运算符：&&（逻辑与），||（逻辑或），!（逻辑非） 账号登录
  + 条件运算符：？ ：
  + 字符串连接运算符：. 作用是将不同的表达式值拼接起来。
  + 错误抑制符：@，抑制错误的显示

### 运算符优先级



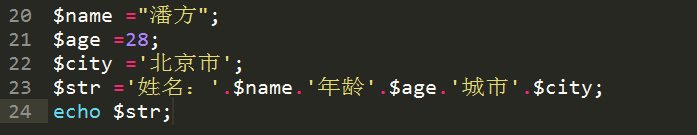


* 算术运算：先乘除后加减。
* 算术运算符 > 比较运算符 > 逻辑运算符 > 赋值运算符
* （）可以改变运算的优先级。

## 特殊运算符

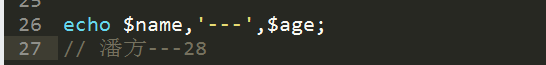
### 字符串连接符 .

遇到比较长的字符串，可以先拼接再输出。PHP有专门的输出缓冲层（output buffer layer），只有缓冲区域的数据达到指定大小，才会输出。尽量使用字符串连接符，提升代码的执行效率。



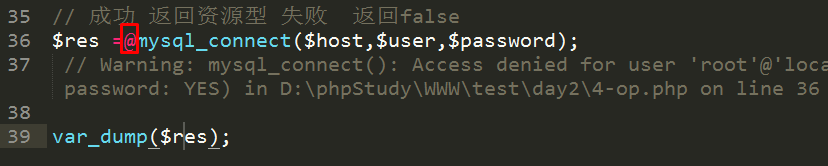
### 字符串分隔符 ,

echo语句可以同时输出多个表达式，多个表达式之间使用英文的逗号分隔。此时将各个表达式作为参数，将echo当作函数使用。



### 错误抑制符 @

抑制错误提示。可以加在变量前面。



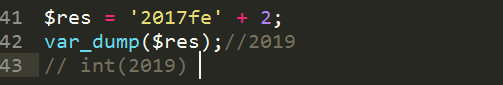
## 算术运算符

+, -, \*, /(除法),%(取余)

6 % 4 =2

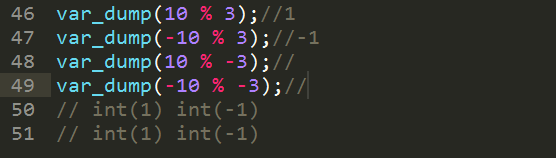
### 计算中的类型自动转换

参与算术运算的所有值，需要转换为数值型，再进行运算。



### 取余运算符%

取余的结果符号取决于被除数的符号。与被除数的符号一致，与除数的符号无关。



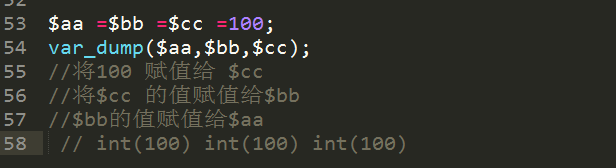
## 赋值运算符=

### 功能

将右侧表达式结果（从右侧到左侧计算）赋值给左侧的变量。

### 赋值运算符的右结合性

* 案例：分析$a =$b =$c =100的执行过程

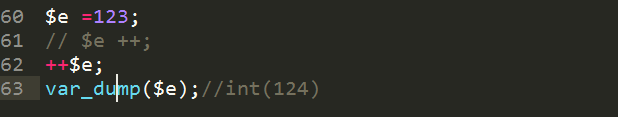


## 自操作运算符

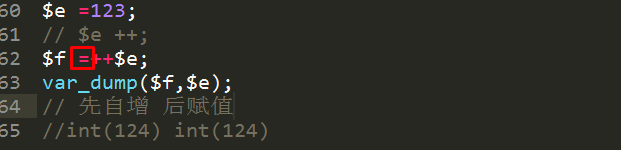
++，--，+=，-=，\*=等

### ++,--两种使用方法

不涉及赋值运算时，自操作运算符在前与在后没有区别。



有赋值运算符的情况：



**先赋值后自增：**



## 比较运算符

>,<,>=,==(相等),!=（不等）,===(全等于),!==（不全等于）

### 相等、全等比较的本质

比较结果只有true或者false。

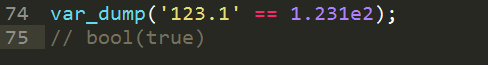


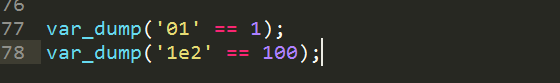
**两个特例：**

* 大于等于 10 >= 9 true
* 布尔值之间的比较 true > false （原因是大小比较时，自动转换为数值类型）

### 相等比较运算原则

不同类型之间进行大小、相等比较，会出现类型的自动转换（不需要类型必须一致）：



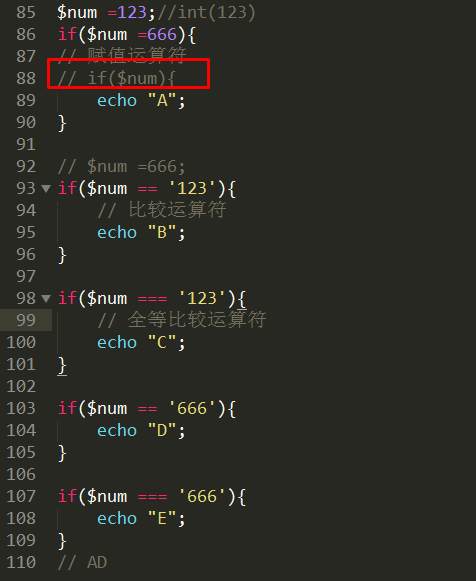


### 全等比较

全等===，不全等！==

全等及不全等，既判断数值，又判断类型。只有类型、值均相等时，才全等。建议判断密码使用全等比较。

### =与==及===的区别



= 赋值，将右侧的表达式的结果赋值给左侧

== 判断是否相等

===判断是否全等

## 逻辑运算符

与（&&）或（||） 非（!）

一般用于条件判断，如使用&&组织多个条件，判断是否同时满足多个条件。如账号登录了，既要判断用户名信息正确还要判断密码信息正确。

### 运算原则

与：全真为真，其余为假

true && true →true

true && false →false

false && false →fasle

或：全假为假，其余为真。

true || true → true

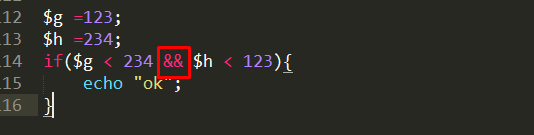
true || false → true

false || false → false

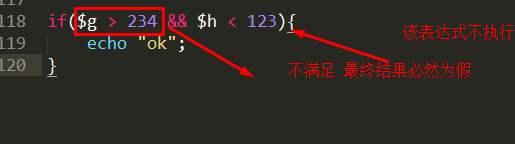
非：非真即假，非假即真

!true → false

!false→ true

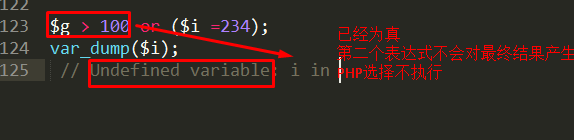


### 短路运算

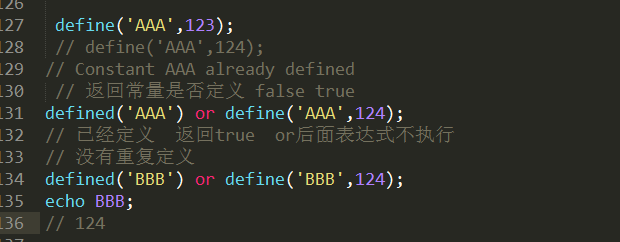


* 或运算的短路运算：

第一个表达式的返回值为true，或后面的表达式不进行判断

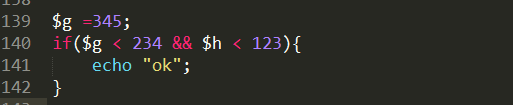


**判断是否已经定义某个常量：**

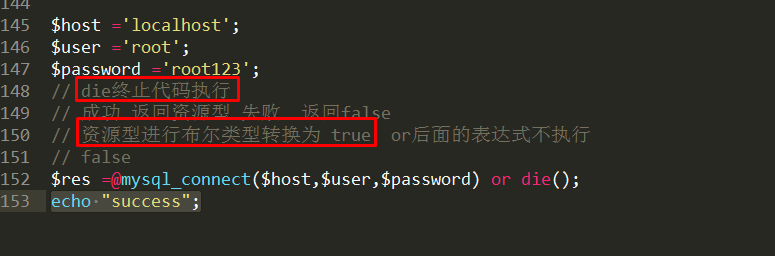


* 与运算的短路运算：

第一个条件为false，进行短路运算。后面的表达式相当于被抛弃。



**短路运算符的应用：**

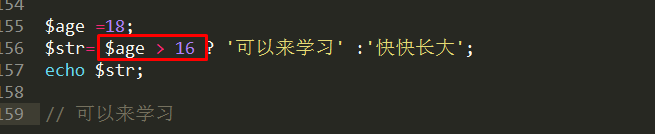


## 条件运算符

又称三元运算符。三个操作数。

条件表达式 ？表达式1 ：表达式2;

条件成立，返回表达式1，否则，返回表达式2.



# 流程控制

程序一般由多个语句组成。流程控制是指按照条件来选择执行的代码或者循环执行一定代码等。流程控制主要用来限定指定代码执行的顺序以及执行的次数。

## 程序语言的三大流程控制

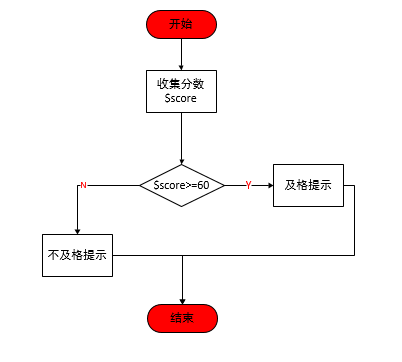
分为顺序、分支（选择）以及循环结构。

## 顺序结构

从上往下顺序、逐行执行。最简单的流程控制。

## 分支（选择结构）

满足指定A分支的条件，执行A分支对应的代码。包括if语句、switch语句。



### 条件执行if else

if(条件){

代码块1；

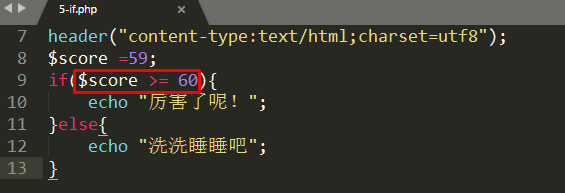
}else{

代码块2；

}

满足条件，执行相应代码。否则执行其他代码。当if语句的分支代码只有一行时，可以省略{}.

* 案例：判断成绩及格与否，及格时，给出及格提示，不及格时，给出不及格提示。



### if elseif else

if(条件1){

分支1；

} elseif (条件2){

分支2；

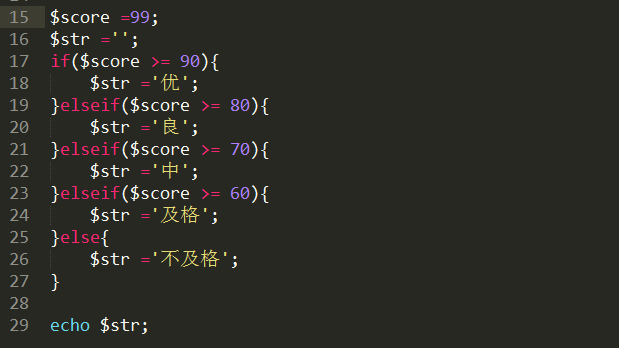
}else{

分支n;

}

实现多分支判断，满足条件，执行对应的语句，否则执行else中的语句。

* 案例：判断成绩，给出成绩等级提示。



### switch语句语法

开关语句。就像有很多开关、很多灯。按某个开关，亮对应的灯。

switch(表达式){

case 值1 ：

分支1；

case 值2 ：

分支2；

….

default :

默认的分支n；

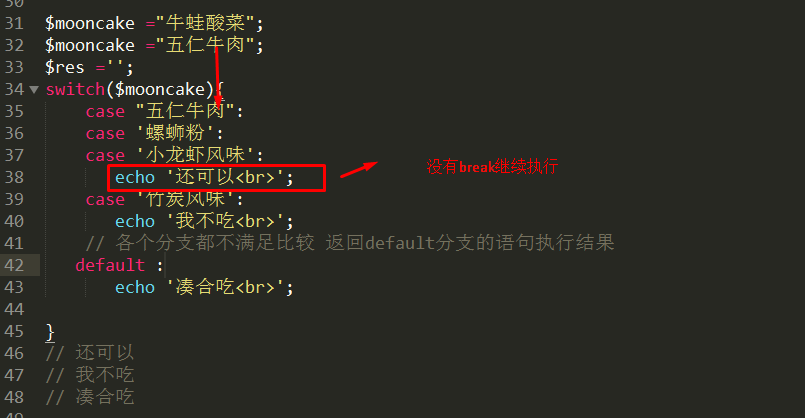
}

使用switch里面的表达式和case的表达式中的值进行比较。相等时，执行相应的语句块。直至最后。

可以添加break语句，实现只执行一个case的语句块。

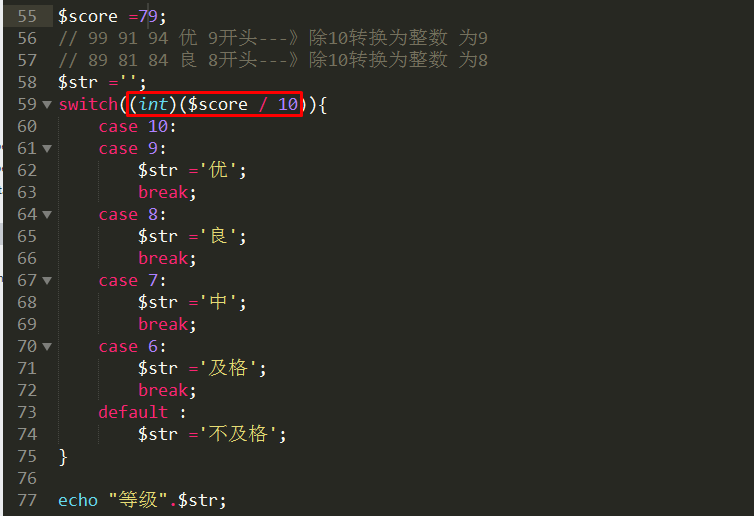
多个case可以共用语句块。

案例：喜欢的月饼--五仁牛肉，牛蛙酸菜，螺蛳粉，竹炭风味，小龙虾风味





案例：成绩等级提示

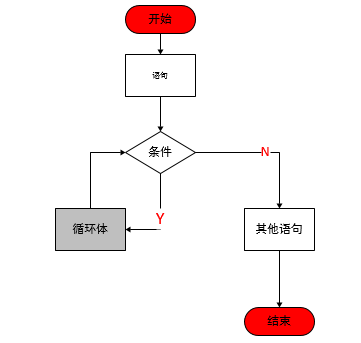


### if与switch区别

if条件判断，一般用来判断是否满足条件。switch语句一般用来做状态(case 的不同值)判断。

## 循环结构

满足一定条件，循环执行指定的代码（循环体）。常见的有for循环, while循环, do-while循环。



### while语法

又称当型循环，当满足指定的条件，循环执行循环体。

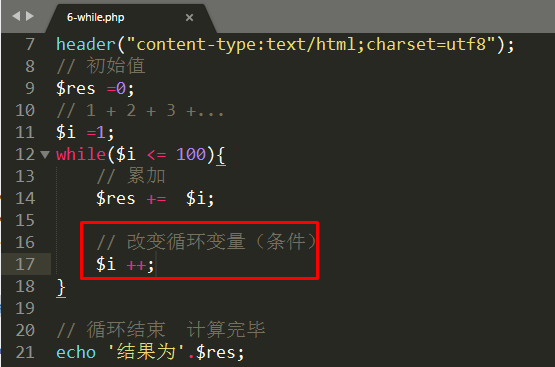
while(循环条件){

循环体；

//改变循环条件，避免陷入死循环

}

案例：计算100之内整数的总和



### do while循环语法

do{

循环体;

}while(条件);

首先执行循环体，再进行判断。当满足条件时，继续执行循环体。不满足，退出循环。

循环体至少执行一次。

**案例：求两个整数的最大公约数。**

**48 24 ：24**

**48 36 ：12**

辗转相除法算法：

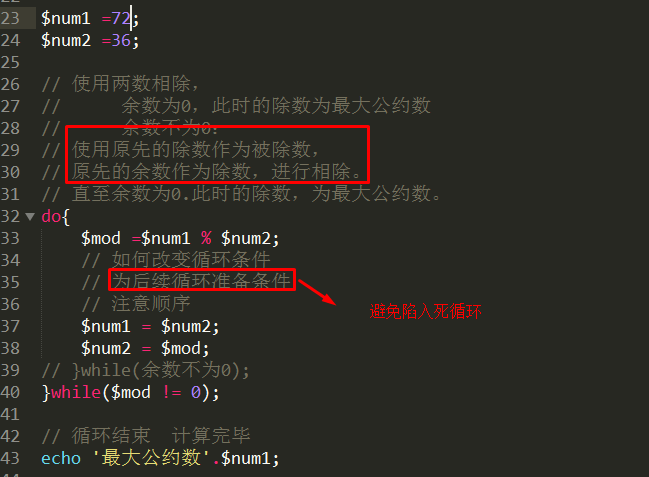
使用两数相除，

* 余数为0，此时的除数为最大公约数
* 余数不为0：

使用原先的除数作为被除数，

原先的余数作为除数，进行相除。

直至余数为0.此时的除数，为最大公约数。



思考：

输出是哪两个数字的余数。

**while与do while区别：**

* while首先判断条件是否满足。满足条件，执行循环体。如果不满足，循环体连一次都不会被执行。
* do-while先执行一次循环，满足条件继续执行循环体。不满足，结束循环。最少执行一次循环体。
* 对于判断条件基于循环体的执行时，必须使用do while.

### for循环语法

for(初始化表达式1[,初始化表达式2]；条件判断;条件变量改变表达式){

循环体；

}

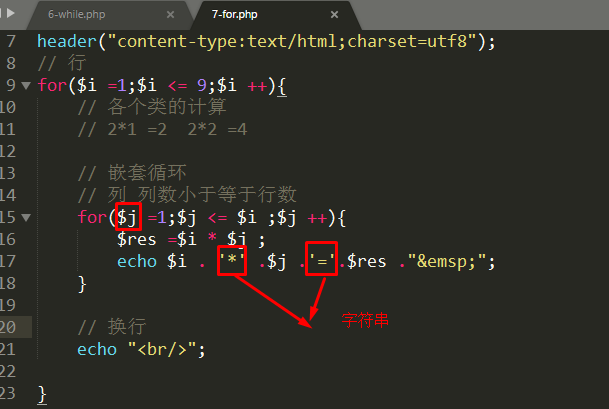
**for执行过程**

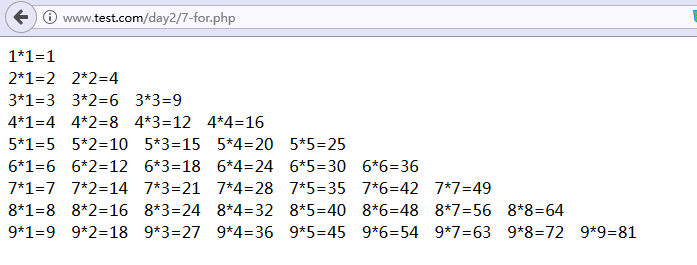
**第一次执行：初始化—条件判断—满足—循环体—条件改变—条件判断**

**案例：九九乘法表**

1\*1 =1

2\*1 =2 2\*2 =4





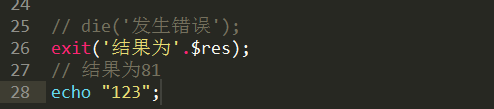
## die,exit

### die,exit

退出代码执行，当前位置之后的所有代码不执行。可以用来进行程序运行控制，可以在出错的时候执行此语句，退出代码的执行。

可以在退出时输出变量值、完成函数执行。

die([…]);



### sleep休眠函数

执行到sleep所在的语句后，延缓一定的秒数，继续执行。可以查看一些临时数据。



# 函数语法

每个页面都需要的、重复的、完成一定功能的代码，如何处理？如格式化打印数组的函数？封装为函数，可以提高代码的复用率。函数是程序中实现特定功能的代码段。

## 声明语法

### 语法

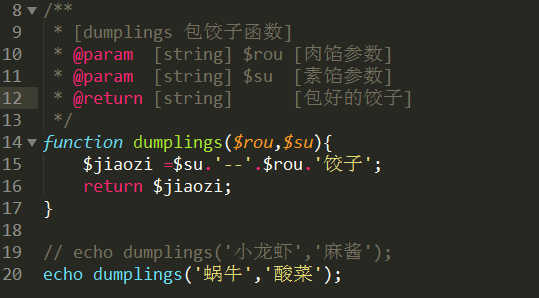
function 函数名（[参数1，参数2]）{

函数体；

//return 语句；

}

### 快速入门：包饺子函数



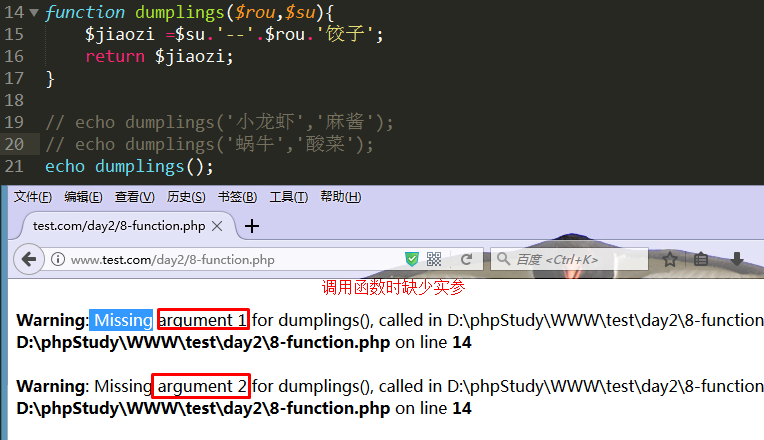
### 五要素

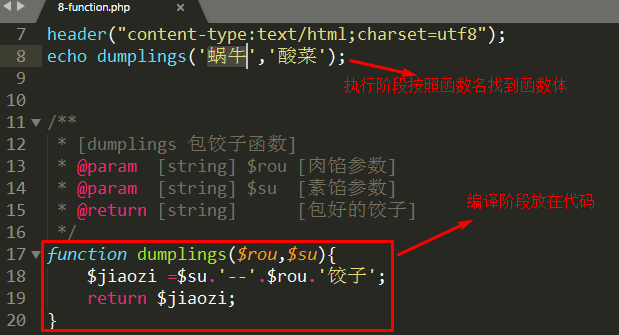
* function关键字。
* 函数名。字母数字下划线组成。不区分大小写。（变量区分，常量区分）
* 参数列表，可选。
* 函数体：函数函数功能的核心代码。
* 返回值。可选。通过return语句实现，将数据返回到调用的地方。

## 调用语法

函数名([参数1，参数2]);

遵循先声明后调用。实际上，调用的位置可以在声明位置之前，不违背先声明后调用的原则。PHP执行的第一阶段为预编译，将函数代码存放在内存的代码区，执行阶段，使用函数名调用函数。



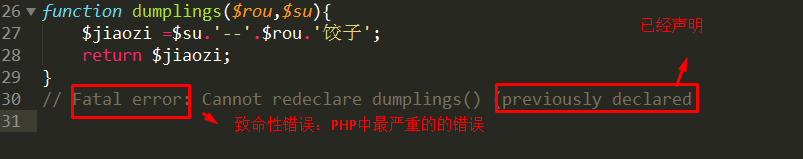


## 函数名命名规则

* 不区分大小写（变量、常量区分大小写）
* 同PHP标识符命名（字母数字下划线组成，不以数字开头）。对于较长的函数名可以使用下划线命名法。

show\_info(); C()

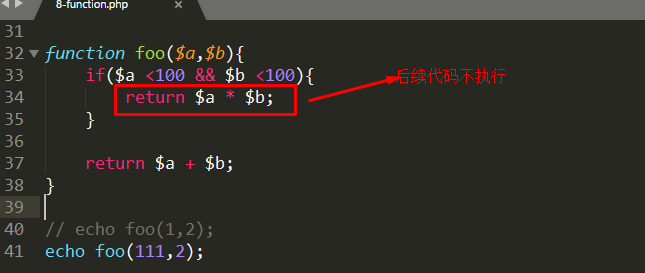
* 函数不能重新定义，不能使用系统函数名。



## 函数体

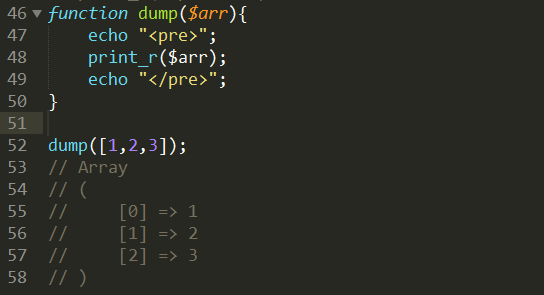
实现函数的功能的代码块。

**return功能：**



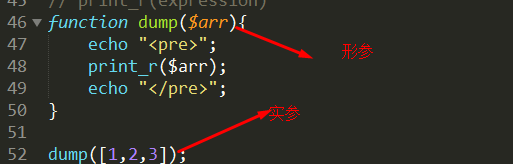
* 将数据返回到调用的地方，调用函数的位置要接收，否则返回值无意义。
* 结束函数的执行，return后面的语句不执行。最多可以执行一个return语句
* return语句不能一次性返回多个值，但可以通过返回一个数组来实现。
* 尽量使用return语句返回信息，而不是直接输出。

案例：返回格式化数组打印的信息



## 函数的参数

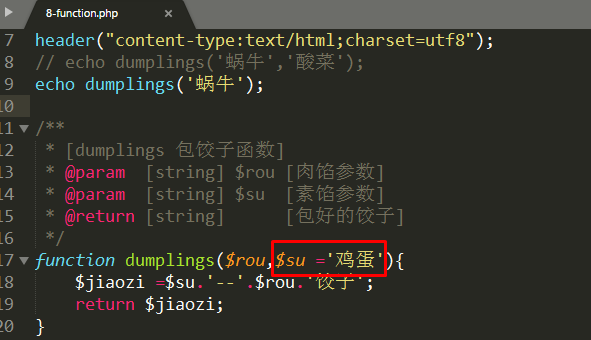
定义函数时可以定义函数的参数，调用时可以将函数的参数传递到函数内部进行加工。函数的参数包括实参及形参。



### 形参

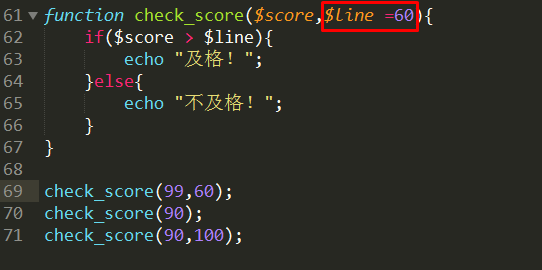
定义时，设置形参参数列表，在函数调用时，将相应的参数传递到函数内部进行加工处理。英文单词：parameter

### 默认值



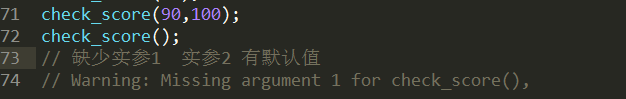
* 参数比较多的情况下，参数值比较固定时，某个参数的值可能不变，可以设置默认值。
* 调用时，没有为有默认值的参数传递参数，则自动使用默认值。设置了形参默认值，同样可以在调用时传递实参。
* 有默认值的形参，该参数尽量靠后设置。
* 默认值不可以是变量

案例:成绩等级查询函数（设置及格线默认为60分）

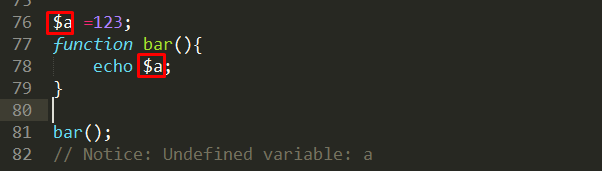


### 实参

调用时将数据传递给对应的形参。英文：argument。实参可以为具体的值，也可以为变量。要与形参的个数一致。



## PHP支持的作用域及特点



函数外部定义的变量，在函数内部无法使用。在函数内部定义的变量，在函数外部无法使用。原因何在？变量的作用域不同

变量的作用域（scope）：变量作用的范围，空间。

### 全局作用域

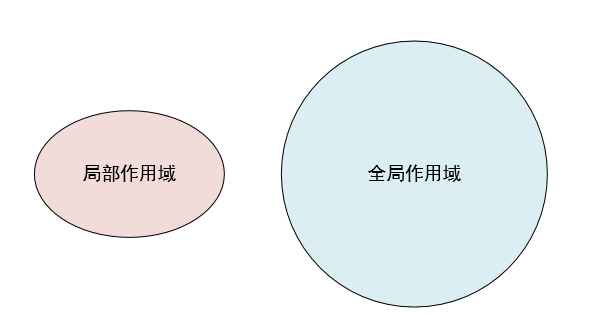
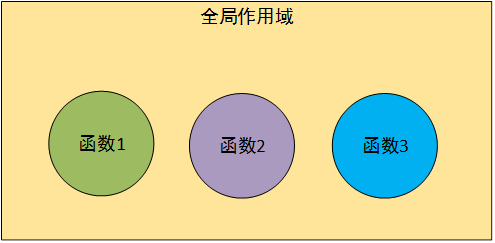
在函数外部定义的变量，为全局作用域内的变量，称为全局变量，只能在全局使用。

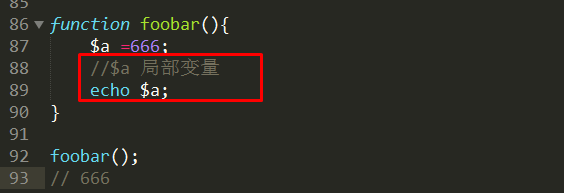
### 局部作用域：

在函数内部定义的变量称为局部变量。变量在函数内部可以使用，外部不可以使用。

### 全局作用域与局部作用域的关系

全局作用域与局部作用域没有交集。不同函数的作用域，也没有交集。

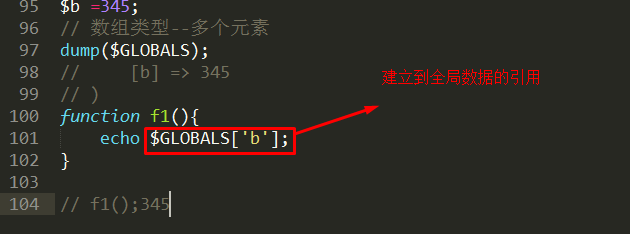
 



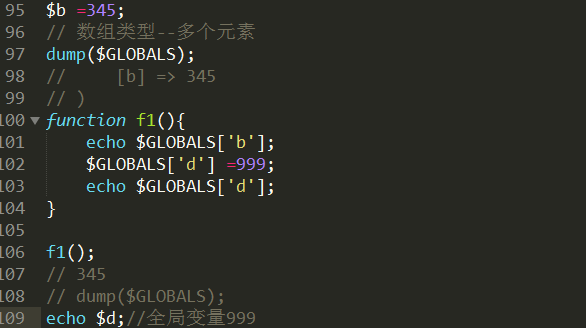
## 超全局变量

如何在局部作用域使用全局变量或者在全局作用域使用局部变量？

$GLOBALS专门用来实现数据的超全局化的超全局变量。作用：引用全局作用域中的全部全局变量。



* 创建一个全局变量， 系统会自动在$GLOBALS中创建变量名为下标的元素。
* 在$GLOBALS中添加元素，自动创建相应名字的全局变量。
* 删除或者修改$GLOBALS数组的元素，会导致全局变量同步修改或删除。



# 系统函数

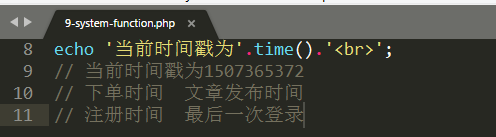
系统预先定义好的函数，可以在任意位置直接调用。

## 时间日期函数

日期时间：年月日 时分秒

### time()

返回自从 Unix 纪元（格林威治时间 1970 年 1 月 1 日 00:00:00）到当前时间的秒数，即时间戳。为10位整型数字。通常用于保存文章的发布时间、商品的发布时间等。



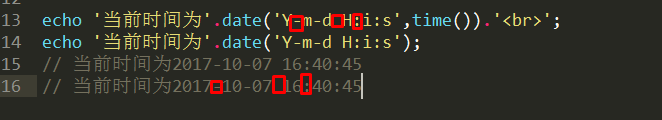
### date()

时间戳为整型数字，无法之间使用，需要先使用的date函数格式化时间戳。

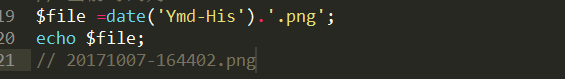
string date ( string $format [, int $timestamp ] )

返回将整数 timestamp 按照给定的格式字串而产生的字符串。如果没有给出时间戳则使用本地当前时间。换句话说，timestamp 是可选的，默认值为 time()。

最常用的格式为：Y-m-d H:i:s （2017-09-13 02:13:14），常用来生成文件名（’Ymd-His’）

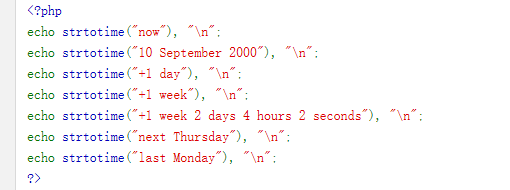


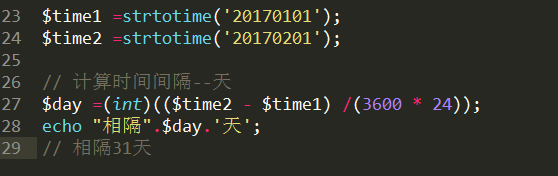




### strtotime()

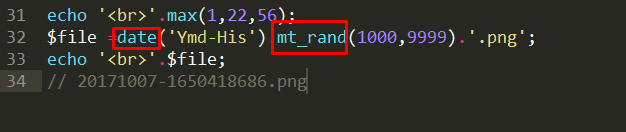
将时间字符串转换为时间戳。





## 数学函数

* max(),min()--分别返回一组数的最大值及最小值
* abs()--返回绝对值。
* floor()--向下取整。
* ceil()--向上取整
* round()--四舍五入。
* rand() 和 mt\_rand()--返回随机数，可以取到两端的值。mt\_rand生成的速度比rand快四倍多。



## 案例：使用数学函数生成验证码文字

decbin():十进制🡪二进制

验证码是打印有随机字符串的图像。随机字符串一般由数字及字母组成，长度一般为4-6。



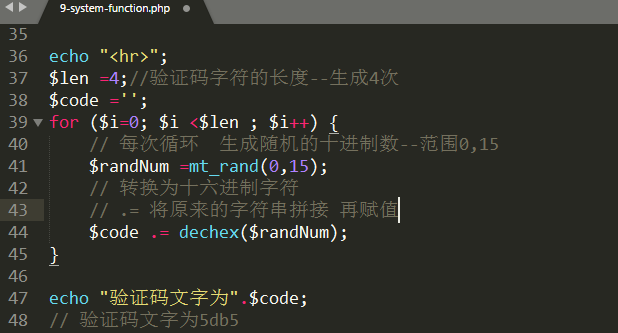
10 –a

4-4

8-8

15-f

生成4个随机的十进制数，转换为对应的十六进制字符，并拼接。



# 文件载入

## 介绍



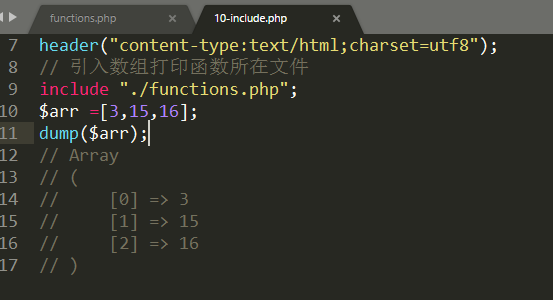
不同的页面中有相同的代码部分，可以将其分离为单个文件。需要调用时，引入对应的文件即可调用。提高代码的复用率。

## 语法

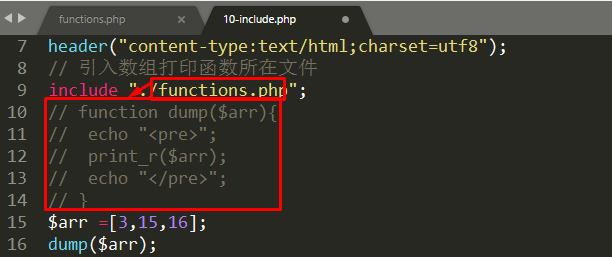
include | include\_once | require |require\_once “文件的路径”

### 原理

将文件代码引入到PHP文件中执行。如果引入的是HTML代码，会在PHP模式与HTML模式之间进行切换。



### 引入PHP文件

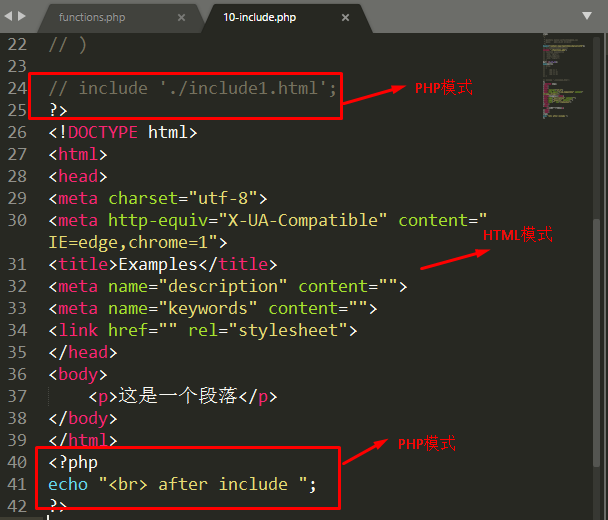


### 引入HTML文件

**原始代码：**



**执行时的等效代码：**



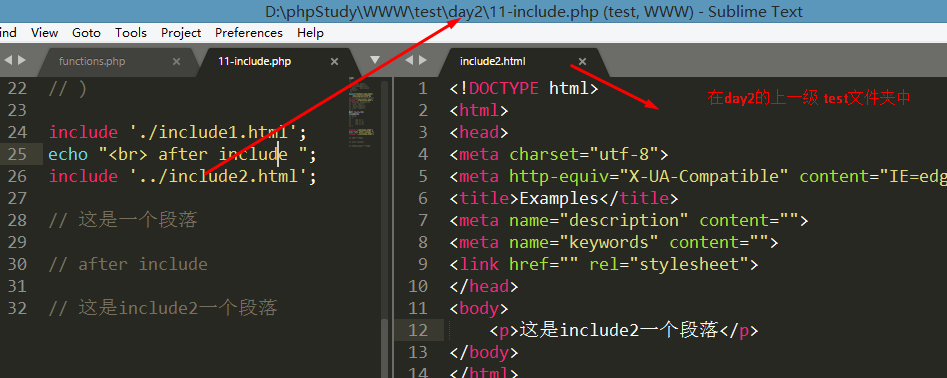
## 路径问题

### 相对路径;

. :表示当前执行脚本所在的目录（文件夹）如include './include1.html'; .对应day2文件夹

.. :当前执行脚本的上一级目录。

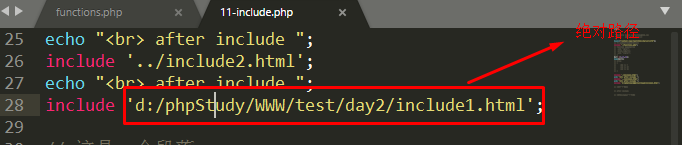
相对于当前脚本所在的路径而言，随执行环境变化而变化。



### 绝对路径

‘/’ 网站根目录

磁盘上文件的真实路径。特点：不便于调试及网站部署。

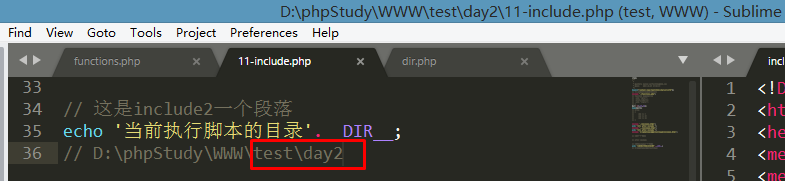


### 魔术常量\_\_DIR\_\_

魔术：随着执行位置变化而变量。

常量：值在编译阶段已经确定。

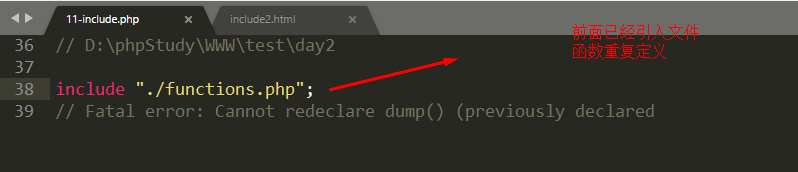
作用：自动获取当前执行脚本的目录。





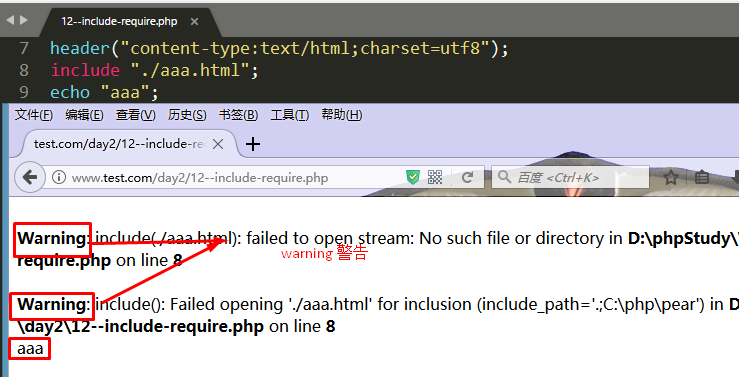
## include与require、include\_once区别

都是语言结构，不是函数。



### include与require区别：

* include引入文件失败，返回警告warning（警告），后续代码继续执行



* require引入文件失败，返回fatal error（致命错误），后续代码不执行



引入的文件为后续代码必须使用的，建议require.

### \_once作用：

对于已经载入文件，不再进行载入，避免函数的重复定义。

