一个字符串 [string](https://www.php.net/manual/zh/language.types.string.php) 就是由一系列的字符组成，其中每个字符等同于一个字节。这意味着 PHP 只能支持 256 的字符集，因此不支持 Unicode 。详见[字符串类型详解](https://www.php.net/manual/zh/language.types.string.php#language.types.string.details)。

**Note**: [string](https://www.php.net/manual/zh/language.types.string.php) 最大可以达到 2GB。

**语法**

一个字符串可以用 4 种方式表达：

* [单引号](https://www.php.net/manual/zh/language.types.string.php#language.types.string.syntax.single)
* [双引号](https://www.php.net/manual/zh/language.types.string.php#language.types.string.syntax.double)
* [heredoc 语法结构](https://www.php.net/manual/zh/language.types.string.php#language.types.string.syntax.heredoc)
* [nowdoc 语法结构](https://www.php.net/manual/zh/language.types.string.php#language.types.string.syntax.nowdoc)（自 PHP 5.3.0 起）

**单引号**

定义一个字符串的最简单的方法是用单引号把它包围起来（字符 *'*）。

要表达一个单引号自身，需在它的前面加个反斜线（*\*）来转义。要表达一个反斜线自身，则用两个反斜线（*\\*）。其它任何方式的反斜线都会被当成反斜线本身：也就是说如果想使用其它转义序列例如 *\r* 或者 *\n*，并不代表任何特殊含义，就单纯是这两个字符本身。

**Note**: 不像[双引号](https://www.php.net/manual/zh/language.types.string.php#language.types.string.syntax.double)和 [heredoc](https://www.php.net/manual/zh/language.types.string.php#language.types.string.syntax.heredoc) 语法结构，在单引号字符串中的[变量](https://www.php.net/manual/zh/language.variables.php)和特殊字符的转义序列将*不会*被替换。

<?php  
echo 'this is a simple string';  
  
// 可以录入多行  
echo 'You can also have embedded newlines in   
strings this way as it is  
okay to do';  
  
// 输出： Arnold once said: "I'll be back"  
echo 'Arnold once said: "I\'ll be back"';  
  
// 输出： You deleted C:\\*.\*?  
echo 'You deleted C:\\\*.\*?';  
  
// 输出： You deleted C:\\*.\*?  
echo 'You deleted C:\\*.\*?';  
  
// 输出： This will not expand: \n a newline  
echo 'This will not expand: \n a newline';  
  
// 输出： Variables do not $expand $either  
echo 'Variables do not $expand $either';  
?>

**双引号**

如果字符串是包围在双引号（"）中， PHP 将对一些特殊的字符进行解析：

| **转义字符** | |
| --- | --- |
| **序列** | **含义** |
| *\n* | 换行（ASCII 字符集中的 LF 或 0x0A (10)） |
| *\r* | 回车（ASCII 字符集中的 CR 或 0x0D (13)） |
| *\t* | 水平制表符（ASCII 字符集中的 HT 或 0x09 (9)） |
| *\v* | 垂直制表符（ASCII 字符集中的 VT 或 0x0B (11)）（自 PHP 5.2.5 起） |
| *\e* | Escape（ASCII 字符集中的 ESC 或 0x1B (27)）（自 PHP 5.4.0 起） |
| *\f* | 换页（ASCII 字符集中的 FF 或 0x0C (12)）（自 PHP 5.2.5 起） |
| *\\* | 反斜线 |
| *\$* | 美元标记 |
| *\"* | 双引号 |
| *\[0-7]{1,3}* | 符合该正则表达式序列的是一个以八进制方式来表达的字符 |
| *\x[0-9A-Fa-f]{1,2}* | 符合该正则表达式序列的是一个以十六进制方式来表达的字符 |

和单引号字符串一样，转义任何其它字符都会导致反斜线被显示出来。PHP 5.1.1 以前，*\{$var}* 中的反斜线还不会被显示出来。

用双引号定义的字符串最重要的特征是变量会被解析，详见[变量解析](https://www.php.net/manual/zh/language.types.string.php#language.types.string.parsing)。

**Heredoc 结构**

第三种表达字符串的方法是用 heredoc 句法结构：*<<<*。在该运算符之后要提供一个标识符，然后换行。接下来是字符串 [string](https://www.php.net/manual/zh/language.types.string.php) 本身，最后要用前面定义的标识符作为结束标志。

结束时所引用的标识符*必须*在该行的第一列，而且，标识符的命名也要像其它标签一样遵守 PHP 的规则：只能包含字母、数字和下划线，并且必须以字母和下划线作为开头。

**Warning**

要注意的是结束标识符这行除了*可能*有一个分号（*;*）外，绝对不能包含其它字符。这意味着标识符*不能缩进*，分号的前后也不能有任何空白或制表符。更重要的是结束标识符的前面必须是个被本地操作系统认可的换行，比如在 UNIX 和 Mac OS X 系统中是 *\n*，而结束定界符（可能其后有个分号）之后也必须紧跟一个换行。

如果不遵守该规则导致结束标识不“干净”，PHP 将认为它不是结束标识符而继续寻找。如果在文件结束前也没有找到一个正确的结束标识符，PHP 将会在最后一行产生一个解析错误。

Heredocs 结构不能用来初始化类的属性。自 PHP 5.3 起，此限制仅对 heredoc 包含变量时有效。

**Example #1 非法的示例**

<?php  
class foo {  
    public $bar = <<<EOT  
bar  
    EOT;  
}  
?>

Heredoc 结构就象是没有使用双引号的双引号字符串，这就是说在 heredoc 结构中单引号不用被转义，但是上文中列出的转义序列还可以使用。变量将被替换，但在 heredoc 结构中含有复杂的变量时要格外小心。

**Example #2 Heredoc 结构的字符串示例**

<?php  
$str = <<<EOD  
Example of string  
spanning multiple lines  
using heredoc syntax.  
EOD;  
  
/\* 含有变量的更复杂示例 \*/  
class foo  
{  
    var $foo;  
    var $bar;  
  
    function foo()  
    {  
        $this->foo = 'Foo';  
        $this->bar = array('Bar1', 'Bar2', 'Bar3');  
    }  
}  
  
$foo = new foo();  
$name = 'MyName';  
  
echo <<<EOT  
My name is "$name". I am printing some $foo->foo.  
Now, I am printing some {$foo->bar[1]}.  
This should print a capital 'A': \x41  
EOT;  
?>

以上例程会输出：

My name is "MyName". I am printing some Foo.

Now, I am printing some Bar2.

This should print a capital 'A': A

也可以把 Heredoc 结构用在函数参数中来传递数据：

**Example #3 Heredoc 结构在参数中的示例**

<?php  
var\_dump(array(<<<EOD  
foobar!  
EOD  
));  
?>

在 PHP 5.3.0 以后，也可以用 Heredoc 结构来初始化静态变量和类的属性和常量：

**Example #4 使用 Heredoc 结构来初始化静态值**

<?php  
// 静态变量  
function foo()  
{  
    static $bar = <<<LABEL  
Nothing in here...  
LABEL;  
}  
  
// 类的常量、属性  
class foo  
{  
    const BAR = <<<FOOBAR  
Constant example  
FOOBAR;  
  
    public $baz = <<<FOOBAR  
Property example  
FOOBAR;  
}  
?>

自 PHP 5.3.0 起还可以在 Heredoc 结构中用双引号来声明标识符：

**Example #5 在 heredoc 结构中使用双引号**

<?php  
echo <<<"FOOBAR"  
Hello World!  
FOOBAR;  
?>

**Nowdoc 结构**

就象 heredoc 结构类似于双引号字符串，Nowdoc 结构是类似于单引号字符串的。Nowdoc 结构很象 heredoc 结构，但是 nowdoc 中*不进行解析操作*。这种结构很适合用于嵌入 PHP 代码或其它大段文本而无需对其中的特殊字符进行转义。与 SGML 的 *<![CDATA[ ]]>* 结构是用来声明大段的不用解析的文本类似，nowdoc 结构也有相同的特征。

一个 nowdoc 结构也用和 heredocs 结构一样的标记 *<<<*， 但是跟在后面的标识符要用单引号括起来，即 *<<<'EOT'*。Heredoc 结构的所有规则也同样适用于 nowdoc 结构，尤其是结束标识符的规则。

**Example #6 Nowdoc 结构字符串示例**

<?php  
$str = <<<'EOD'  
Example of string  
spanning multiple lines  
using nowdoc syntax.  
EOD;  
  
/\* 含有变量的更复杂的示例 \*/  
class foo  
{  
    public $foo;  
    public $bar;  
  
    function foo()  
    {  
        $this->foo = 'Foo';  
        $this->bar = array('Bar1', 'Bar2', 'Bar3');  
    }  
}  
  
$foo = new foo();  
$name = 'MyName';  
  
echo <<<'EOT'  
My name is "$name". I am printing some $foo->foo.  
Now, I am printing some {$foo->bar[1]}.  
This should not print a capital 'A': \x41  
EOT;  
?>

以上例程会输出：

My name is "$name". I am printing some $foo->foo.

Now, I am printing some {$foo->bar[1]}.

This should not print a capital 'A': \x41

**Note**:

不象 heredoc 结构，nowdoc 结构可以用在任意的静态数据环境中，最典型的示例是用来初始化类的属性或常量：

**Example #7 静态数据的示例**

<?php  
class foo {  
    public $bar = <<<'EOT'  
bar  
EOT;  
}  
?>

**Note**:

Nowdoc 结构是在 PHP 5.3.0 中加入的。

**变量解析**

当字符串用双引号或 heredoc 结构定义时，其中的[变量](https://www.php.net/manual/zh/language.variables.php)将会被解析。

这里共有两种语法规则：一种[简单](https://www.php.net/manual/zh/language.types.string.php#language.types.string.parsing.simple)规则，一种[复杂](https://www.php.net/manual/zh/language.types.string.php#language.types.string.parsing.complex)规则。简单的语法规则是最常用和最方便的，它可以用最少的代码在一个 [string](https://www.php.net/manual/zh/language.types.string.php) 中嵌入一个变量，一个 [array](https://www.php.net/manual/zh/language.types.array.php) 的值，或一个 [object](https://www.php.net/manual/zh/language.types.object.php) 的属性。

复杂规则语法的显著标记是用花括号包围的表达式。

**简单语法**

当 PHP 解析器遇到一个美元符号（*$*）时，它会和其它很多解析器一样，去组合尽量多的标识以形成一个合法的变量名。可以用花括号来明确变量名的界线。

<?php  
$juice = "apple";  
  
echo "He drank some $juice juice.".PHP\_EOL;  
// Invalid. "s" is a valid character for a variable name, but the variable is $juice.  
echo "He drank some juice made of $juices.";  
?>

以上例程会输出：

He drank some apple juice.

He drank some juice made of .

类似的，一个 [array](https://www.php.net/manual/zh/language.types.array.php) 索引或一个 [object](https://www.php.net/manual/zh/language.types.object.php) 属性也可被解析。数组索引要用方括号（*]*）来表示索引结束的边际，对象属性则是和上述的变量规则相同。

**Example #8 简单语法示例**

<?php  
$juices = array("apple", "orange", "koolaid1" => "purple");  
  
echo "He drank some $juices[0] juice.".PHP\_EOL;  
echo "He drank some $juices[1] juice.".PHP\_EOL;  
echo "He drank some juice made of $juice[0]s.".PHP\_EOL; // Won't work  
echo "He drank some $juices[koolaid1] juice.".PHP\_EOL;  
  
class people {  
    public $john = "John Smith";  
    public $jane = "Jane Smith";  
    public $robert = "Robert Paulsen";  
      
    public $smith = "Smith";  
}  
  
$people = new people();  
  
echo "$people->john drank some $juices[0] juice.".PHP\_EOL;  
echo "$people->john then said hello to $people->jane.".PHP\_EOL;  
echo "$people->john's wife greeted $people->robert.".PHP\_EOL;  
echo "$people->robert greeted the two $people->smiths."; // Won't work  
?>

以上例程会输出：

He drank some apple juice.

He drank some orange juice.

He drank some juice made of s.

He drank some purple juice.

John Smith drank some apple juice.

John Smith then said hello to Jane Smith.

John Smith's wife greeted Robert Paulsen.

Robert Paulsen greeted the two .

如果想要表达更复杂的结构，请用复杂语法。

**复杂（花括号）语法**

复杂语法不是因为其语法复杂而得名，而是因为它可以使用复杂的表达式。

任何具有 [string](https://www.php.net/manual/zh/language.types.string.php) 表达的标量变量，数组单元或对象属性都可使用此语法。只需简单地像在 [string](https://www.php.net/manual/zh/language.types.string.php) 以外的地方那样写出表达式，然后用花括号 *{* 和 *}* 把它括起来即可。由于 *{* 无法被转义，只有 *$* 紧挨着 *{* 时才会被识别。可以用 *{\$* 来表达 *{$*。下面的示例可以更好的解释：

<?php  
// 显示所有错误  
error\_reporting(E\_ALL);  
  
$great = 'fantastic';  
  
// 无效，输出: This is { fantastic}  
echo "This is { $great}";  
  
// 有效，输出： This is fantastic  
echo "This is {$great}";  
echo "This is ${great}";  
  
// 有效  
echo "This square is {$square->width}00 centimeters broad.";   
  
// 有效，只有通过花括号语法才能正确解析带引号的键名  
echo "This works: {$arr['key']}";  
  
// 有效  
echo "This works: {$arr[4][3]}";  
  
// 这是错误的表达式，因为就象 $foo[bar] 的格式在字符串以外也是错的一样。  
// 换句话说，只有在 PHP 能找到常量 foo 的前提下才会正常工作；这里会产生一个  
// E\_NOTICE (undefined constant) 级别的错误。  
echo "This is wrong: {$arr[foo][3]}";   
  
// 有效，当在字符串中使用多重数组时，一定要用括号将它括起来  
echo "This works: {$arr['foo'][3]}";  
  
// 有效  
echo "This works: " . $arr['foo'][3];  
  
echo "This works too: {$obj->values[3]->name}";  
  
echo "This is the value of the var named $name: {${$name}}";  
  
echo "This is the value of the var named by the return value of getName(): {${getName()}}";  
  
echo "This is the value of the var named by the return value of \$object->getName(): {${$object->getName()}}";  
  
// 无效，输出： This is the return value of getName(): {getName()}  
echo "This is the return value of getName(): {getName()}";  
?>

也可以在字符串中用此语法通过变量来调用类的属性。

<?php  
class foo {  
    var $bar = 'I am bar.';  
}  
  
$foo = new foo();  
$bar = 'bar';  
$baz = array('foo', 'bar', 'baz', 'quux');  
echo "{$foo->$bar}\n";  
echo "{$foo->{$baz[1]}}\n";  
?>

以上例程会输出：

I am bar.  
I am bar.

**Note**:

函数、方法、静态类变量和类常量只有在 PHP 5 以后才可在 *{$}* 中使用。然而，只有在该字符串被定义的命名空间中才可以将其值作为变量名来访问。只单一使用花括号 (*{}*) 无法处理从函数或方法的返回值或者类常量以及类静态变量的值。

<?php  
// 显示所有错误  
error\_reporting(E\_ALL);  
  
class beers {  
    const softdrink = 'rootbeer';  
    public static $ale = 'ipa';  
}  
  
$rootbeer = 'A & W';  
$ipa = 'Alexander Keith\'s';  
  
// 有效，输出： I'd like an A & W  
echo "I'd like an {${beers::softdrink}}\n";  
  
// 也有效，输出： I'd like an Alexander Keith's  
echo "I'd like an {${beers::$ale}}\n";  
?>

**存取和修改字符串中的字符**

[string](https://www.php.net/manual/zh/language.types.string.php) 中的字符可以通过一个从 0 开始的下标，用类似 [array](https://www.php.net/manual/zh/language.types.array.php) 结构中的方括号包含对应的数字来访问和修改，比如 *$str[42]*。可以把 [string](https://www.php.net/manual/zh/language.types.string.php) 当成字符组成的 [array](https://www.php.net/manual/zh/language.types.array.php)。函数 [substr()](https://www.php.net/manual/zh/function.substr.php) 和 [substr\_replace()](https://www.php.net/manual/zh/function.substr-replace.php) 可用于操作多于一个字符的情况。

**Note**: [string](https://www.php.net/manual/zh/language.types.string.php) 也可用花括号访问，比如 *$str{42}*。

**Warning**

用超出字符串长度的下标写入将会拉长该字符串并以空格填充。非整数类型下标会被转换成整数。非法下标类型会产生一个 **E\_NOTICE** 级别错误。用负数下标写入字符串时会产生一个 **E\_NOTICE** 级别错误，用负数下标读取字符串时返回空字符串。写入时只用到了赋值字符串的第一个字符。用空字符串赋值则赋给的值是 NULL 字符。

**Warning**

PHP 的字符串在内部是字节组成的数组。因此用花括号访问或修改字符串对多字节字符集很不安全。仅应对单字节编码例如 ISO-8859-1 的字符串进行此类操作。

**Example #9 一些字符串示例**

<?php  
// 取得字符串的第一个字符  
$str = 'This is a test.';  
$first = $str[0];  
  
// 取得字符串的第三个字符  
$third = $str[2];  
  
// 取得字符串的最后一个字符  
$str = 'This is still a test.';  
$last = $str[strlen($str)-1];   
  
// 修改字符串的最后一个字符  
$str = 'Look at the sea';  
$str[strlen($str)-1] = 'e';  
  
?>

自 PHP 5.4 起字符串下标必须为整数或可转换为整数的字符串，否则会发出警告。之前类似 *"foo"* 的下标会无声地转换成 *0*。

**Example #10 PHP 5.3 和 PHP 5.4 的区别**

<?php  
$str = 'abc';  
  
var\_dump($str['1']);  
var\_dump(isset($str['1']));  
  
var\_dump($str['1.0']);  
var\_dump(isset($str['1.0']));  
  
var\_dump($str['x']);  
var\_dump(isset($str['x']));  
  
var\_dump($str['1x']);  
var\_dump(isset($str['1x']));  
?>

以上例程在PHP 5.3中的输出：

string(1) "b"

bool(true)

string(1) "b"

bool(true)

string(1) "a"

bool(true)

string(1) "b"

bool(true)

以上例程在PHP 5.4中的输出：

string(1) "b"

bool(true)

Warning: Illegal string offset '1.0' in /tmp/t.php on line 7

string(1) "b"

bool(false)

Warning: Illegal string offset 'x' in /tmp/t.php on line 9

string(1) "a"

bool(false)

string(1) "b"

bool(false)

**Note**:

用 *[]* 或 *{}* 访问任何其它类型（不包括数组或具有相应接口的对象实现）的变量只会无声地返回 **NULL**。

**Note**:

PHP 5.5 增加了直接在字符串原型中用 *[]* 或 *{}* 访问字符的支持。

**有用的函数和运算符**

字符串可以用 '.'（点）运算符连接起来，注意 '+'（加号）运算符*没有*这个功能。更多信息参考[字符串运算符](https://www.php.net/manual/zh/language.operators.string.php)。

对于 [string](https://www.php.net/manual/zh/language.types.string.php) 的操作有很多有用的函数。

可以参考[字符串函数](https://www.php.net/manual/zh/ref.strings.php)了解大部分函数，高级的查找与替换功能可以参考[正则表达式函数](https://www.php.net/manual/zh/ref.regex.php)或 [Perl 兼容正则表达式函数](https://www.php.net/manual/zh/ref.pcre.php)。

另外还有 [URL 字符串函数](https://www.php.net/manual/zh/ref.url.php)，也有加密／解密字符串的函数（[mcrypt](https://www.php.net/manual/zh/ref.mcrypt.php) 和 [mhash](https://www.php.net/manual/zh/ref.mhash.php)）。

最后，可以参考[字符类型函数](https://www.php.net/manual/zh/ref.ctype.php)。

**转换成字符串**

一个值可以通过在其前面加上 *(string)* 或用 [strval()](https://www.php.net/manual/zh/function.strval.php) 函数来转变成字符串。在一个需要字符串的表达式中，会自动转换为 [string](https://www.php.net/manual/zh/language.types.string.php)。比如在使用函数 [echo](https://www.php.net/manual/zh/function.echo.php) 或 [print](https://www.php.net/manual/zh/function.print.php) 时，或在一个变量和一个 [string](https://www.php.net/manual/zh/language.types.string.php) 进行比较时，就会发生这种转换。[类型](https://www.php.net/manual/zh/language.types.php)和[类型转换](https://www.php.net/manual/zh/language.types.type-juggling.php)可以更好的解释下面的事情，也可参考函数 [settype()](https://www.php.net/manual/zh/function.settype.php)。

一个布尔值 [boolean](https://www.php.net/manual/zh/language.types.boolean.php) 的 **TRUE** 被转换成 [string](https://www.php.net/manual/zh/language.types.string.php) 的 *"1"*。[Boolean](https://www.php.net/manual/zh/language.types.boolean.php) 的 **FALSE** 被转换成 *""*（空字符串）。这种转换可以在 [boolean](https://www.php.net/manual/zh/language.types.boolean.php) 和 [string](https://www.php.net/manual/zh/language.types.string.php) 之间相互进行。

一个整数 [integer](https://www.php.net/manual/zh/language.types.integer.php) 或浮点数 [float](https://www.php.net/manual/zh/language.types.float.php) 被转换为数字的字面样式的 [string](https://www.php.net/manual/zh/language.types.string.php)（包括 [float](https://www.php.net/manual/zh/language.types.float.php) 中的指数部分）。使用指数计数法的浮点数（*4.1E+6*）也可转换。

**Note**:

在脚本的区域（category LC\_NUMERIC）中定义了十进制小数点字符。参见 [setlocale()](https://www.php.net/manual/zh/function.setlocale.php)。

数组 [array](https://www.php.net/manual/zh/language.types.array.php) 总是转换成字符串 *"Array"*，因此，[echo](https://www.php.net/manual/zh/function.echo.php) 和 [print](https://www.php.net/manual/zh/function.print.php) 无法显示出该数组的内容。要显示某个单元，可以用 *echo $arr['foo']* 这种结构。要显示整个数组内容见下文。

在 PHP 4 中对象 [object](https://www.php.net/manual/zh/language.types.object.php) 总是被转换成字符串 *"Object"*，如果为了调试原因需要打印出对象的值，请继续阅读下文。为了得到对象的类的名称，可以用 [get\_class()](https://www.php.net/manual/zh/function.get-class.php) 函数。自 PHP 5 起，适当时可以用 [\_\_toString](https://www.php.net/manual/zh/language.oop5.magic.php) 方法。

资源 [resource](https://www.php.net/manual/zh/language.types.resource.php) 总会被转变成 *"Resource id #1"* 这种结构的字符串，其中的 *1* 是 PHP 在运行时分配给该 [resource](https://www.php.net/manual/zh/language.types.resource.php) 的唯一值。不要依赖此结构，可能会有变更。要得到一个 [resource](https://www.php.net/manual/zh/language.types.resource.php) 的类型，可以用函数 [get\_resource\_type()](https://www.php.net/manual/zh/function.get-resource-type.php)。

**NULL** 总是被转变成空字符串。

如上面所说的，直接把 [array](https://www.php.net/manual/zh/language.types.array.php)，[object](https://www.php.net/manual/zh/language.types.object.php) 或 [resource](https://www.php.net/manual/zh/language.types.resource.php) 转换成 [string](https://www.php.net/manual/zh/language.types.string.php) 不会得到除了其类型之外的任何有用信息。可以使用函数 [print\_r()](https://www.php.net/manual/zh/function.print-r.php) 和 [var\_dump()](https://www.php.net/manual/zh/function.var-dump.php) 列出这些类型的内容。

大部分的 PHP 值可以转变成 [string](https://www.php.net/manual/zh/language.types.string.php) 来永久保存，这被称作串行化，可以用函数 [serialize()](https://www.php.net/manual/zh/function.serialize.php) 来实现。如果 PHP 引擎设定支持 [WDDX](https://www.php.net/manual/zh/ref.wddx.php)，PHP 值也可被串行化为格式良好的 XML 文本。

**字符串转换为数值**

当一个字符串被当作一个数值来取值，其结果和类型如下：

如果该字符串没有包含 '.'，'e' 或 'E' 并且其数字值在整型的范围之内（由 **PHP\_INT\_MAX** 所定义），该字符串将被当成 [integer](https://www.php.net/manual/zh/language.types.integer.php) 来取值。其它所有情况下都被作为 [float](https://www.php.net/manual/zh/language.types.float.php) 来取值。

该字符串的开始部分决定了它的值。如果该字符串以合法的数值开始，则使用该数值。否则其值为 0（零）。合法数值由可选的正负号，后面跟着一个或多个数字（可能有小数点），再跟着可选的指数部分。指数部分由 'e' 或 'E' 后面跟着一个或多个数字构成。

<?php  
$foo = 1 + "10.5";                // $foo is float (11.5)  
$foo = 1 + "-1.3e3";              // $foo is float (-1299)  
$foo = 1 + "bob-1.3e3";           // $foo is integer (1)  
$foo = 1 + "bob3";                // $foo is integer (1)  
$foo = 1 + "10 Small Pigs";       // $foo is integer (11)  
$foo = 4 + "10.2 Little Piggies"; // $foo is float (14.2)  
$foo = "10.0 pigs " + 1;          // $foo is float (11)  
$foo = "10.0 pigs " + 1.0;        // $foo is float (11)       
?>

更多信息可以参考 Unix 手册中的 strtod(3)。

本节中的示例可以通过复制／粘贴到下面的代码中来显示：

<?php  
echo "\$foo==$foo; type is " . gettype ($foo) . "<br />\n";  
?>

不要想像在 C 语言中的那样，通过将一个字符转换成整数以得到其代码。使用函数 [ord()](https://www.php.net/manual/zh/function.ord.php) 和 [chr()](https://www.php.net/manual/zh/function.chr.php) 实现 ASCII 码和字符间的转换。

**字符串类型详解**

PHP 中的 [string](https://www.php.net/manual/zh/language.types.string.php) 的实现方式是一个由字节组成的数组再加上一个整数指明缓冲区长度。并无如何将字节转换成字符的信息，由程序员来决定。字符串由什么值来组成并无限制；特别的，其值为 *0*（“NUL bytes”）的字节可以处于字符串任何位置（不过有几个函数，在本手册中被称为非“二进制安全”的，也许会把 NUL 字节之后的数据全都忽略）。

字符串类型的此特性解释了为什么 PHP 中没有单独的“byte”类型 - 已经用字符串来代替了。返回非文本值的函数 - 例如从网络套接字读取的任意数据 - 仍会返回字符串。

由于 PHP 并不特别指明字符串的编码，那字符串到底是怎样编码的呢？例如字符串 *"á"* 到底是等于 *"\xE1"*（ISO-8859-1），*"\xC3\xA1"*（UTF-8，C form），*"\x61\xCC\x81"*（UTF-8，D form）还是任何其它可能的表达呢？答案是字符串会被按照该脚本文件相同的编码方式来编码。因此如果一个脚本的编码是 ISO-8859-1，则其中的字符串也会被编码为 ISO-8859-1，以此类推。不过这并不适用于激活了 Zend Multibyte 时；此时脚本可以是以任何方式编码的（明确指定或被自动检测）然后被转换为某种内部编码，然后字符串将被用此方式编码。注意脚本的编码有一些约束（如果激活了 Zend Multibyte 则是其内部编码）- 这意味着此编码应该是 ASCII 的兼容超集，例如 UTF-8 或 ISO-8859-1。不过要注意，依赖状态的编码其中相同的字节值可以用于首字母和非首字母而转换状态，这可能会造成问题。

当然了，要做到有用，操作文本的函数必须假定字符串是如何编码的。不幸的是，PHP 关于此的函数有很多变种：

* 某些函数假定字符串是以单字节编码的，但并不需要将字节解释为特定的字符。例如 [substr()](https://www.php.net/manual/zh/function.substr.php)，[strpos()](https://www.php.net/manual/zh/function.strpos.php)，[strlen()](https://www.php.net/manual/zh/function.strlen.php) 和 [strcmp()](https://www.php.net/manual/zh/function.strcmp.php)。理解这些函数的另一种方法是它们作用于内存缓冲区，即按照字节和字节下标操作。
* 某些函数被传递入了字符串的编码方式，也可能会假定默认无此信息。例如 [htmlentities()](https://www.php.net/manual/zh/function.htmlentities.php) 和 [mbstring](https://www.php.net/manual/zh/book.mbstring.php) 扩展中的大部分函数。
* 其它函数使用了当前区域（见 [setlocale()](https://www.php.net/manual/zh/function.setlocale.php)），但是逐字节操作。例如 [strcasecmp()](https://www.php.net/manual/zh/function.strcasecmp.php)，[strtoupper()](https://www.php.net/manual/zh/function.strtoupper.php) 和 [ucfirst()](https://www.php.net/manual/zh/function.ucfirst.php)。这意味着这些函数只能用于单字节编码，而且编码要与区域匹配。例如 *strtoupper("á")* 在区域设定正确并且 *á* 是单字节编码时会返回 *"Á"*。如果是用 UTF-8 编码则不会返回正确结果，其结果根据当前区域有可能返回损坏的值。
* 最后一些函数会假定字符串是使用某特定编码的，通常是 UTF-8。[intl](https://www.php.net/manual/zh/book.intl.php) 扩展和 [PCRE](https://www.php.net/manual/zh/book.pcre.php)（上例中仅在使用了 *u* 修饰符时）扩展中的大部分函数都是这样。尽管这是由于其特殊用途，[utf8\_decode()](https://www.php.net/manual/zh/function.utf8-decode.php) 会假定 UTF-8 编码而 [utf8\_encode()](https://www.php.net/manual/zh/function.utf8-encode.php) 会假定 ISO-8859-1 编码。

最后，要书写能够正确使用 Unicode 的程序依赖于很小心地避免那些可能会损坏数据的函数。要使用来自于 [intl](https://www.php.net/manual/zh/book.intl.php) 和 [mbstring](https://www.php.net/manual/zh/book.mbstring.php) 扩展的函数。不过使用能处理 Unicode 编码的函数只是个开始。不管用何种语言提供的函数，最基本的还是了解 Unicode 规格。例如一个程序如果假定只有大写和小写，那可是大错特错。