

UD2 - Inserción de código en páxinas web



Comezando a traballar con PHP



Índice

2.1 Introducción

2.2 Xeración de código HTML

2.3 Comentarios

2.4 Variables

2.5 Tipos de datos

2.6 Constantes

2.7 Expresións e operadores

2.8 Funcións

2.9 Ámbitos de uso das variables





2.1 Introducción

- Ademais de permitir empregar comandos para amosar por pantalla código HTML (como CGI ou Perl), as páxinas de PHP conteñen HTML con código incrustado que se executará para realizar aquello que se lle indique. No seguinte exemplo o que fai é amosar por pantalla o texto “*Ola chavalotes/as!!*”. O código incrustase coas etiquetas especiais `<?php ?>` ou acurtadas `<?>` que indican que se vai entrar ou saír do código PHP e segundo esteamos ou non nese modo se entenderá que é código de cliente ((X)HTML, JS, CSS) ou código de servidor (PHP).

The screenshot shows a code editor window titled "PHP Code". The tab bar at the top has "PHP" selected. The code editor displays the following HTML structure:

```
<html>
  <head></head>
  <body>
    <h1>PHP</h1>
    <?php echo '<h2>Ola chavalotes/as!!</h2>'; ?>
    </body>
  </html>
```

A green rectangular box highlights the PHP code block: `<?php echo '<h2>Ola chavalotes/as!!</h2>'; ?>`. A green arrow points from the text "Escrito en PHP" to this highlighted block.



2.1 Introducción

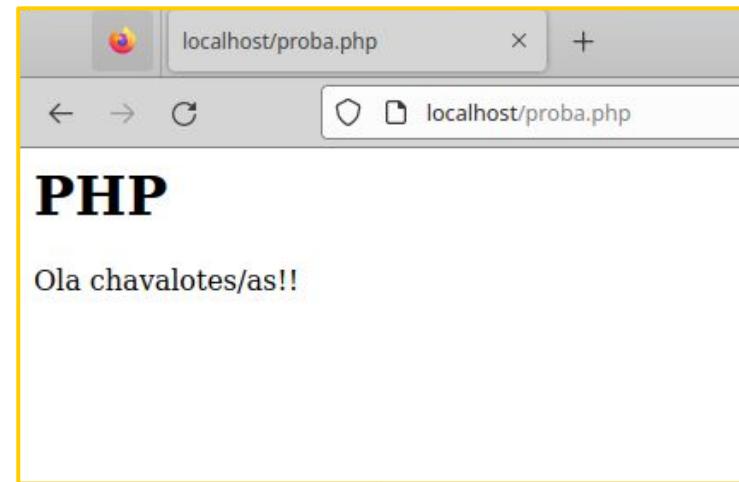
- O que se envía dende o servidor ao cliente (navegador do usuario) é o código HTML resultante (engadindo CSS e JS).

PHP code rendered as HTML

HTML

```
<html>
  <head></head>
  <body>
    <h1>PHP</h1>
    <h2>Ola chavalotes/as!!</h2>
  </body>
</html>
```

Convértese en HTML





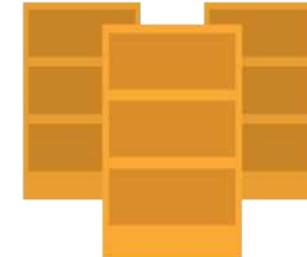
2.1 Introducción

User types in URL

1 `HTTP://SOMEAWESOMEWEBSITE.COM`

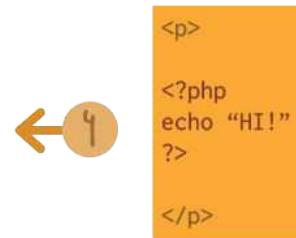
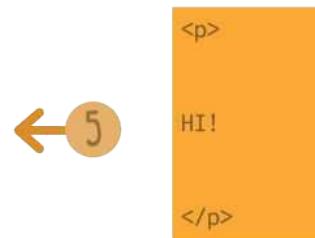
Browser requests page from server

2



3

Server executes
PHP code within
requested file



Final page is sent back to
the browser

5

4

Result of PHP code is inserted into
the rest of the page

5



2.1 Introducción

- Os **punto e coma** permiten separar diferentes sentenzas en PHP en secuencia:

```
index.php x

<?php
    // Comentario
    echo 'Hello';          Punto e coma
    echo 'Learn PHP!';     Punto e coma
?
>
```

- Podes gardar un código similar ao do exemplo nun ficheiro de texto no teu servidor web en </var/www/html> con, por exemplo, co nome <ola.php> e acceder ao mesmo dende un navegador <http://localhost/ola.php>.
- A recomendación PSR-1 di que os documentos PHP deben usar a **codificación UTF-8 sen BOM**.



2.2 Xeración de código HTML

- En moitas linguaxes de programación acostúmase a emplegar funcións para amosar texto por pantalla (no navegador).
- En PHP isto significa converter o código PHP a HTML.
- Pódense enviar tanto texto como etiquetas HTML, código CSS ou Javascript.

Imprimir cadeas de texto ou Strings

index.php

```
<?php echo ' Ola chavalotes/as!! ' ; ?>
```

Imprime o texto entre comiñas no navegador

Preview

Ola chavalotes/as!!

Os Strings ou cadeas de texto se indican con comiñas dobles " ou simples '

index.php

```
<?php echo "Hello, World!"; ?>
```

// Output: Hello, World!

```
<?php echo 'Hello, PHP!'; ?>
```

// Output: Hello, PHP!

- Unha función habitual para amosar texto ou enviar código HTML ao navegador é a función **echo()**.



2.2 Xeración de código HTML

- Na función `echo()` tanto as comiñas simples como as dobles permiten definir cadeas de caracteres pero vanse comportar de maneira diferente. As dobles van permitir incluír variables directamente na cadea de texto.

```
index.php  x

$name = 'Ken the Ninja';
echo "Hello, {$name}";
// Output: Hello, Ken the Ninja
echo 'Hello, {$name}';
// Output: Hello,{\$name}

As comiñas dobles permiten
reemplazar o valor de $name

Coas simples interpreta todo como string
```

The screenshot shows a code editor window titled "index.php". It contains the following PHP code:

```
$name = 'Ken the Ninja';
echo "Hello, {$name}";
// Output: Hello, Ken the Ninja
echo 'Hello, {$name}';
// Output: Hello,{\$name}
```

Annotations in green text are overlaid on the code:

- "As comiñas dobles permiten reemplazar o valor de \$name" is positioned above the first echo statement, explaining the output "Hello, Ken the Ninja".
- "Coas simples interpreta todo como string" is positioned below the second echo statement, explaining the output "Hello,{\\$name}".

Nota: As chaves {} son opcionais.



2.2 Xeración de código HTML

Existen varias formas de inserir contido na páxina web a partir do resultado da execución de código PHP:

- "Función" **echo**. Que non devolve nada (void) e xera como saída o texto dos parámetros que recibe. Non precisa parénteses.
 - `void echo(string $arg, ...)`
- "Función" **print**. Que funciona de xeito similar, pero só recibe un parámetro e devolve sempre o valor 1. Non precisa parénteses.
 - `int print(string $arg)`
- Función **printf()**. Pode recibir varios parámetros. O primeiro será unha cadea de texto que indica o formato a aplicar, esa cadea debe conter un especificador de conversión por cada parámetro que se lle pase á función. Devolve o tamaño da cadea.
 - `int printf (string $format, [$arg1], ...)`
- Funcións **die()** e **exit()**. Amosan a mensaxe que recibe como argumento e deteñen o script.



```
printf("Amosa unha cadea que pode recibir parámetros %s %d", "cadea", 1)
die("Depuración: Detén o script");
```



2.2 Xeración de código HTML

Secuencias de escape

- Determinados caracteres especiais dentro das cadeas de texto poden causar conflitos.
- As secuencias de escape permiten indicar que ese caracteres ou símbolos especiais deben ser tratados como texto ou caracteres normais.
- Só nas cadeas que se definan con comiñas dobres.
- Caracteres conflitivos: “, ‘, \$, e a barra \ cando defina outros caracteres especiais como \t, \f, etc, polo que habería que facer \\.

```
<?php  
  
// Erros de análise:  
$cadea = 'D'Artagnan';  
$cadea = "Jon dixo \"achégase o inverno\".";  
$ruta = "c:\test\ficheiro.txt";  
  
?>
```

```
<?php  
  
// Correctas:  
$cadea = 'D\'Artagnan';  
$cadea = "Jon dixo \'achégase o inverno\'.";  
$ruta = "c:\\test\\ficheiro.txt";  
  
?>
```

```
$mensaxe = "Exemplo de cadea de texto: \"Ola, isto non é unha \$variable\"";
```



2.2 Xeración de código HTML

Formas de xerar código HTML con PHP

```
● ● ●  
<?php  
// Isto é PHP  
$texto = "1,2,3, ... probando ";  
?  
  
<!-- Isto é HTML -->  
<html>  
<body>  
  <div>  
    <h1><?php echo $texto; ?></h1>  
  </div>  
</body>  
</html>
```

A boa

```
● ● ●  
<?php  
// Isto é PHP  
$texto = "1,2,3, ... probando ";  
  
$html = "<html><body><div><h1>" . $texto .  
"</h1></div></body></html>";  
echo $html;  
?>
```

A mala

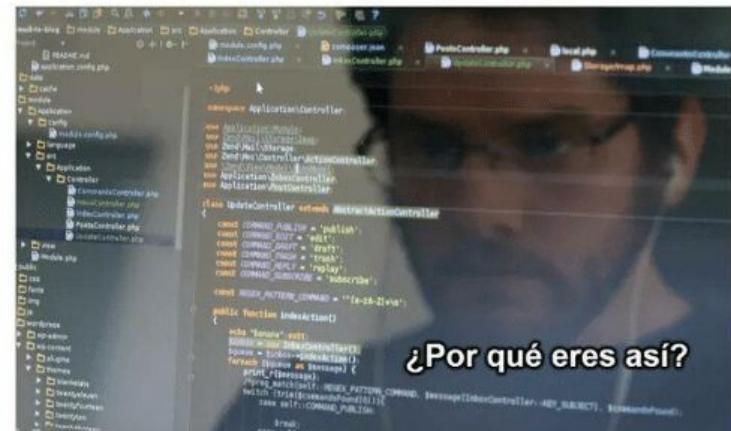


2.3 Comentarios

Como en todas as linguaxes de programación en PHP podemos inserir **comentarios ou textos explicativos de como funciona un programa ou script**. Estes textos axudan a entender como funciona o programa e aclaran determinados aspectos que sobre todo a longo prazo aforran moito traballo. Dispoñemos de varios tipos de comentarios:

- **Dentro do HTML**, delimitados por `<!--` e `-->`. Estes comentarios son enviados dentro da páxina web e **poderán velos os usuarios**, cosa que non vai acontecer cos comentarios en PHP.
- **Dentro do PHP** das seguintes formas:
 - Comentarios de liña empregando `//`
→ Considérase o estándar de facto (recomendado)
 - Comentarios de liña empregando `#`
→ Só en versións recentes de PHP (non recomendado)
 - Comentarios de varias liñas cos delimitadores `/*` e `*/`.

Cuando sabes que debes comentar tu código pero nunca lo haces y al final no entiendes lo que programaste



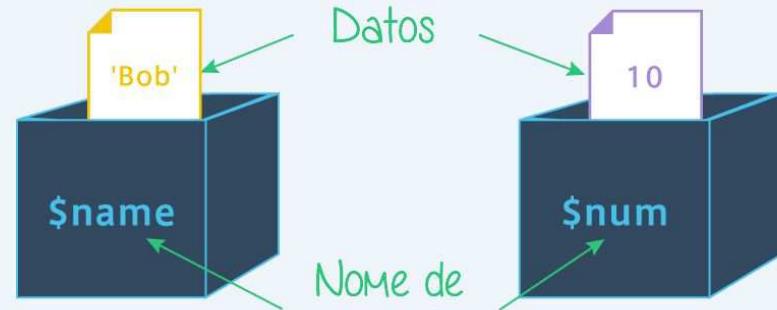
Nota: Os comentarios en PHP execútanse no servidor e non se converten a HTML polo que **non poden velos o usuarios**.



2.4 Variables

- Dende PHP pódense crear **variables** para **almacenar valores que cambien** ao longo da execución do script ou programa.
- Sempre deben comezar polo signo **\$**.
- A diferenza de noutras linguaxes de programación moitas linguaxes de script, entre elas PHP, non precisan que se declare a variable nin se defina o tipo de datos (enteiro, cadea de texto, etc).
- Simplemente haberá que asignarlle o primeiro valor co operador de asignación **=** e nese momento colerá o tipo de datos dese valor.

O concepto de variables



`$name = 'Bob';`

`$number = 10;`



```
// Exemplo #1 Variables
<?php
    $contador = 1; // Ao asignarolle un 1, a variable é de tipo enteiro
    $texto = "un"; // Ao asignarolle un 'un', a variable é de tipo cadea de texto
?>
```



2.4 Variables

Recordatorios para o uso de variables

Á hora de definir variables, cómpre ter en conta o seguinte:

- As variables de PHP deben **comezar por** un símbolo de **dólar (\$)**.
- Despois do símbolo de dólar deben levar unha letra ou un guión baixo (_), nunca números.
- As variables só admiten **caracteres alfanuméricos e guións baixos** (a-z, A-Z, 0-9 ou _).
- **Distinguen entre maiúsculas e minúsculas**, polo que `$varSuma` non é o mesmo que `$VarSuma`.
- Sempre hai que tentar seguir a mesma **nomenclatura**.



camelCase



kebab-case



snake_case



2.5 Tipos de datos

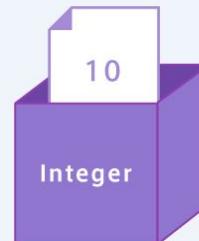
- Os **tipos de datos** simples en PHP serían:
 - **Booleano** (boolean). Valores true ou false. Ademais, calquera número enteiro considérase como true, salvo o 0 que é false.
 - **Enteiro** (integer). Calquera número sen decimais, positivo ou negativo.
 - **Real** (float o double). Calquera número con decimais.
 - **Cadea de texto** (string). Conxuntos de caracteres alfanuméricos delimitados por comiñas simples o dobles.
- **Nulo (NULL/null)**. Indica que a variable non ten valor.

Tipos de datos



Tipos de texto:

- Caracteres, palabras, frases, etc



Tipos numéricos:

- Enteros (-1, 0, 1, 2, ...)
- Reais (3.14, 1.1618, ...)



2.5 Tipos de datos

LENGUAJES DE PROGRAMACIÓN TIPADOS VS NO TIPADOS

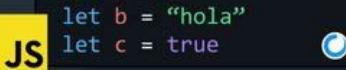
Los lenguajes estáticamente tipados requieren declarar las **variables** con **su tipo de dato**.

```
int a = 1;
String b = "hola";
bool c = true;
```



Los lenguajes dinámicamente tipados declaran **variables** sin **necesidad de definir el tipo de dato** (el intérprete infiere el tipo).

```
let a = 1
let b = "hola"
let c = true
```



Más **verboso**, pero menos propenso a errores de **sintaxis** (el compilador los detecta).

Se puede saber qué tipo de dato retorna una función.

```
func saludar() string {
```



No se puede cambiar el tipo de dato después de declarada la variable.



Código más legible y curva de aprendizaje más sencilla.

No sabes qué tipo de dato retorna una función.

```
def saludar():
```



Se puede cambiar el tipo de dato después de declarada la variable.



Tipos dinámicos

ou

Debilmente tipados



2.5 Tipos de datos

- No caso de realizar operacións entre variables de distintos tipos ambas convértense ao mesmo tipo común. Por exemplo, sumar un enteiro a un real resulta nun real. Concatenar unha cadea de texto e un enteiro da lugar a unha cadea de texto. Aínda que se realizan de xeito automático tamén se poden forzar:



Exemplo #2 Conversión de tipos

```
<?php
$foo = "0"; // $foo é un string (ASCII 48)
$foo += 2; // $foo pasou a ser un enteiro (2)
$foo = $foo + 1.3; // $foo é agora un real (3.3)
$foo = 5 + "10 porquiños pequeñiños"; // $foo é enteiro(15)
$foo = 10; // $foo é un enteiro
$bar = (boolean) $foo; // $bar é un boolean
var_dump($bar);
?>
```

Nota: función [var_dump\(\)](#) amosa información sobre unha variable incluíndo o seu tipo e o seu valor.



2.5 Tipos de datos

- Dispoñemos dunha serie de **funcións xa definidas para relacionadas co tipo de datos**, que permiten comprobar e establecer os tipos de datos das variables.
- Función **gettype()** obtén o tipo dunha variable que se lle pasa como parámetro e devolve unha cadea de texto: **array, boolean, double, integer, object, string, null, resource** ou **unknown type**.
- Tamén podemos comprobar **se a variable é dun tipo concreto** utilizando unha das seguintes funcións: **is_array(), is_bool(), is_float(), is_integer(), is_null(), is_numeric(), is_object(), is_resource(), is_scalar() e is_string()**, que devuelven true / verdadeiro se a variable é do tipo indicado.
- Analogamente, para establecer o tipo dunha variable se emprega a función **settype()** pasándolle como parámetros a variable a converter, e unha das seguintes cadeas: **boolean, integer, float, string, array, object** ou **null**. A función devolve true / verdadeiro se a conversión se realizou correctamente, ou false / falso en caso contrario.

```
1 <?php
2 $a = $b = "3.1416"; // asignamos ás dúas variables o mesmo String
3 settype($b, "float"); // e cambiamos $b a tipo float (real)
4 print "\$a vale $b e é de tipo ".gettype($b);
5 ?>
```

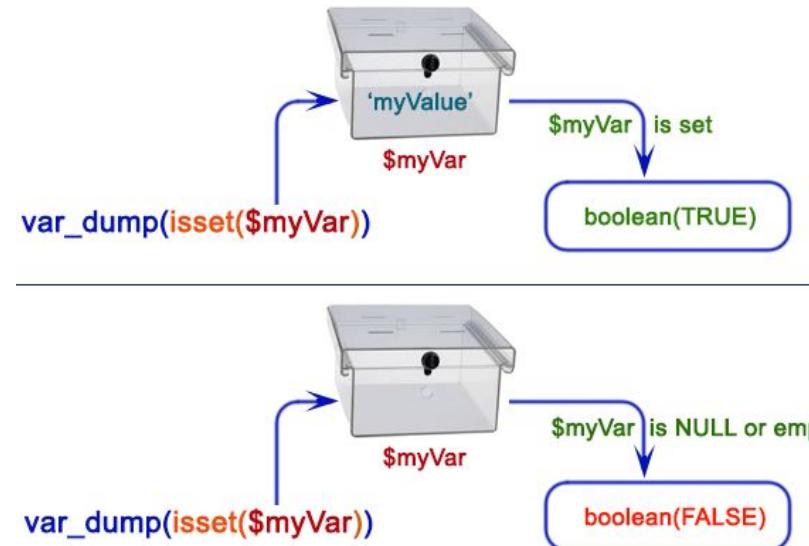
Que devolverá a cadea: \$a vale 3.1416 e é de tipo double (real)

Nota: En PHP non existe diferenza entre os tipos de datos float/double ou real. Todos se almacenan como **double**, de aí a mensaxe anterior.



2.5 Tipos de datos

- Se o único que se precisa é saber se unha variable está definida e non é **null**, pode usar a función **isset()**.
- A función **unset()** destrúe a variable ou variables que se lle pasan como parámetro.
- É importante non confundir que unha variable estea definida ou teña o valor **null**, con que se considere como baleira debido ao valor que contén. Isto último é o que nos indica a función **empty()**. Devolve **false** se a variable existe e ten un valor non baleiro, distinto de cero. Doutro modo devolve **true**. As seguintes expresións son consideradas como baleiras:
 - \$var; (unha variable declarada, pero sen un valor ou sobre a que se fixo **unset()**)
 - "" (String ou cadea baleira)
 - 0 (0 coma un *integer*)
 - 0.0 (0 coma un *float*)
 - "0" (0 coma un *string*)
 - NULL
 - FALSE
 - array() (un array baleiro)





2.6 Constantes

Constantes (I)

As constantes son valores permanentes invariables durante a execución dun programa ou script. Para facer máis flexible o código acostuman a identificarse con letras maiúsculas e utilizando o guión baixo como separador.

```
Exemplo #1 Constantes (I)

<?php
    define('ANO', '2034');
    echo "<p>Estamos no ano " . ANO . "</p>";
?>
<p></p>

<?php
    printf("Ola. Estamos no ano %s. Comezou en Domingo e fixose o cambio
de moeda oficial do euro ao Dogecoin.", ANO);
?>
```



2.6 Constantes

Constantes (II)

- A función printf() permite definir parámetros que sexan ocupados por variables ou constantes.
- Dentro da propia constante tamén podemos definir parámetros que despois sexan substituídos polos seus valores coas funcións printf() e sprintf().



Exemplo #2 Constantes (II)

```
<?php
    define('NOME', 'Naruto Uzumaki');
    printf(' Ola %s', NOME);

    define('CADEA_PRESENTACION', 'Ola usuario/a %s');
    printf(CADEA_PRESENTACION, 'Barristan Selmy');

?>
```



2.6 Constantes

Constantes (III)

PHP dispón de constantes predefinidas coma por exemplo:

- `true` e `false`
- `null`
- `PHP_INT_SIZE`, tamaño en bytes dun enteiro no contorno actual (por exemplo 8 bytes a partires de PHP 7).
- `PHP_INT_MAX`, máximo enteiro soportado.
- `PHP_VERSION`, versión actual de PHP.

Exemplo #3 Constantes (III)

```
<?php
    $tam = PHP_INT_SIZE;
    printf("<p>Os enteiros gárdanse con $tam bytes</p>");
?>
```



2.7 Expresións e operadores

En PHP unha expresión é unha combinación de valores, variables, operadores e/ou funcións. Pódense utilizar as expresións para realizar accións dentro dun programa ou script. Todas as **expresións** deben conter coma mínimo un **operando** e un **operador**. Por exemplo:

```
$foo = 7;  
$a = $b + $c;  
$valor++;  
$x += 5;
```

```
index.php  x  
  
echo 7 + 3; // Output: 10  
echo 10 - 4; // Output: 6  
echo 3 * 6; // Output: 18  
echo 9 / 3; // Output: 3  
echo 5 % 2; // Output: 1
```



2.7 Expresións e operadores

Os operadores permiten:

- Realizar operacións aritméticas: negación (un menos diante devolve o oposto ou negativo), suma, resta, multiplicación, división, módulo e expoñente (**). Entre estes inclúense operadores de pre e post incremento e decremento, `++` e `--`.
`$a = 5;`
`$b = ++$a; // $a e $b pasan a ter valor 6.`
- Realizar asignacións. Ademais do operador `=`, existen operadores cos que realizar operacións e asignacións nun único paso (`+=`, `-=`,...).

The screenshot shows a code editor window titled "index.php". It displays two columns of PHP code side-by-side, comparing standard assignment operators with their abbreviated counterparts. The "Estándar" column on the left contains five lines of code, each consisting of a variable assignment followed by an arithmetic operator and a value. The "Abreviado" column on the right shows the equivalent code using the shorthand assignment operators. The code editor has a dark theme with syntax highlighting for PHP keywords and operators.

Estándar	Abreviado
<code>\$x = \$x + 10;</code>	<code>\$x += 10;</code>
<code>\$x = \$x - 10;</code>	<code>\$x -= 10;</code>
<code>\$x = \$x * 10;</code>	<code>\$x *= 10;</code>
<code>\$x = \$x / 10;</code>	<code>\$x /= 10;</code>
<code>\$x = \$x % 10;</code>	<code>\$x %= 10;</code>



2.7 Expresións e operadores

- **Comparar** operandos. Ademais dos que nos podemos atopar noutros linguaxes (`>`, `>=`, ...), en PHP temos dous operadores para comprobar igualdade (`==`, `===`) e tres para comprobar diferenza (`<>`, `!=` e `!==`). Os operadores `<>` e `!=` son equivalentes. O operador `==` devolve verdadeiro (true) só se os operandos son do mesmo tipo e ademais teñen o mesmo valor. Por exemplo:

```
$x = 0;
```

```
/* $x == false é verdadeira (devolve true), pero $x === false non é certa (devolve false) pois $x  
é de tipo enteiro, non booleano */
```

Ejemplo	Nombre	Resultado
<code>\$a == \$b</code>	Igual	<code>true</code> si <code>\$a</code> es igual a <code>\$b</code> después de la manipulación de tipos.
<code>\$a === \$b</code>	Idéntico	<code>true</code> si <code>\$a</code> es igual a <code>\$b</code> , y son del mismo tipo.
<code>\$a != \$b</code>	Diferente	<code>true</code> si <code>\$a</code> no es igual a <code>\$b</code> después de la manipulación de tipos.
<code>\$a <> \$b</code>	Diferente	<code>true</code> si <code>\$a</code> no es igual a <code>\$b</code> después de la manipulación de tipos.
<code>\$a !== \$b</code>	No idéntico	<code>true</code> si <code>\$a</code> no es igual a <code>\$b</code> , o si no son del mismo tipo.
<code>\$a < \$b</code>	Menor que	<code>true</code> si <code>\$a</code> es estrictamente menor que <code>\$b</code> .
<code>\$a > \$b</code>	Mayor que	<code>true</code> si <code>\$a</code> es estrictamente mayor que <code>\$b</code> .
<code>\$a <= \$b</code>	Menor o igual que	<code>true</code> si <code>\$a</code> es menor o igual que <code>\$b</code> .
<code>\$a >= \$b</code>	Mayor o igual que	<code>true</code> si <code>\$a</code> es mayor o igual que <code>\$b</code> .
<code>\$a <> \$b</code>	Nave espacial	Un integer menor que, igual a, o maior que cero cuando <code>\$a</code> es respectivamente menor que, igual a, o maior que <code>\$b</code> . Disponible a partir de PHP 7.
<code>\$a ?? \$b ?? \$c</code>	Fusión de null	El primer operando de izquierda a derecha que exista y no sea <code>null</code> . <code>null</code> si no hay valores definidos y no son <code>null</code> . Disponible a partir de PHP 7.



2.7 Expresións e operadores

- **Comparacións lóxicas** para expresións booleanas. Tratan aos operandos coma variables booleanas (true ou false). Existen operadores para realizar un E lóxico (operadores **AND** ou **&&**), OU lóxico (operadores **OR** ou **||**), Non lóxico (operador **!**) e o OU lóxico exclusivo (operador **XOR**), será certo se calquera dos dous operandos o é pero falso se os dous o son.

Ejemplo	Nombre	Resultado
<code>\$a and \$b</code>	And (y)	true si tanto \$a como \$b son true.
<code>\$a or \$b</code>	Or (o inclusivo)	true si cualquiera de \$a o \$b es true.
<code>\$a xor \$b</code>	Xor (o exclusivo)	true si \$a o \$b es true, pero no ambos.
<code>! \$a</code>	Not (no)	true si \$a no es true.
<code>\$a && \$b</code>	And (y)	true si tanto \$a como \$b son true.
<code>\$a \$b</code>	Or (o inclusivo)	true si cualquiera de \$a o \$b es true.



A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0



2.7 Expresións e operadores

- **Operacións con cadeas de texto.** Existen dous operadores para datos tipo string. O primeiro é o operador de concatenación ('.'), o cal devolve o resultado de concatenar os seus argumentos dereito e esquierdo. O segundo é o operador de asignación sobre concatenación ('.='), o cal engade o argumento do lado dereito ao do lado esquierdo.

The screenshot shows a code editor window titled "index.php". The code contains the following PHP script:

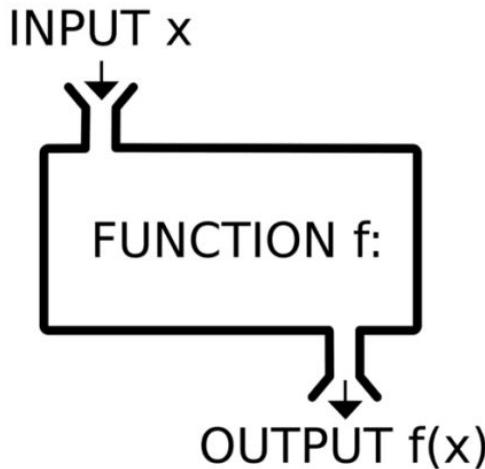
```
$name = 'Ken ';
$name .= 'the Ninja';
echo $name;
// Output: Ken the Ninja
```

A green callout box highlights the line `$name .= 'the Ninja';`. A green bracket below it indicates that this is equivalent to `$name = $name . "the Ninja";`.



2.8 Funcións

- Cando se quere **repetir a execución dun código** en diferentes partes dun programa pódese emplegar unha función. PHP contén infinitade de **funcións predefinidas**, pero ademais pódense definir **funcións a medida**.
- As funcións asocian unha etiqueta co nome da función co bloque de código que se vai executar. As funcións permiten estruturar moito mellor o código e reutilizalo de xeito doado.
- As variables que se empregan nas funcións son de ámbito local, polo que non serán visibles fóra das mesmas. Ademais, as funcións reciben unha serie de valores na forma de parámetros ou argumentos e devuelven un valor, a saída da función.



index.php

```
echo strlen( 'Rodolfo' );
```

Argumento

Función que devolve o tamaño dunha cadea de texto ou String

// Saída: 7



2.8 Funcións

- Para definir unha función propia hai que emplegar a sentenza `function nome_da_función()` seguida do código da mesma.

Definindo unha función

```
index.php  x  
  
function Ola () {  
    echo 'Ola chavalotes/as!!';  
}
```

```
Ola ();
```

 () requiridos ao chamar a unha función

// Saída: Ola Chavalotes/as !!



2.8 Funcións

¿Para que nos serven?

Se temos que repetir código en diferentes lugares podemos agrupalo para definilo unha soa vez e utilizalo as veces que faga falta.

Calculando a área dun círculo

index.php

```
$radius1 = 3;  
echo $radius1 * $radius1 * 3;  
  
$radius2 = 5;  
echo $radius2 * $radius2 * 3;
```

Faise o mesmo nos dous sitios

index.php

Definindo a función

```
function printCircleArea($radius) {  
    echo $radius * $radius * 3;  
}
```

Chamando á función

```
printCircleArea(3); // Saída : 27  
printCircleArea(5); // Saída : 75
```



2.8 Funcións

Argumentos

A unha función pódenselle pasar unha serie de valores, chamados **argumentos**. Estes deben definirse coma **parámetros** á hora de definir a función.

Definindo unha función

index.php x

```
function printSum($num1, $num2) {  
    echo $num1 + $num2;  
}
```

Parámetros

Chamando á función

index.php x

```
printSum(1, 3);  
// Saída : 4
```

Estes argumentos
pásanse aos parámetros

```
function printSum($num1, $num2) {  
    echo $num1 + $num2;  
}
```



2.8 Funcións

Valores a devolver

- As funcións sempre devolven un valor indicado coa sentenza `return valor` que indica o valor que vai devolver a función, aínda que é opcional.
- No caso de que se omita esa sentenza leva implícito que o valor que devolve a función é o valor `NULL`, de tipo `void`. Un exemplo serían as funcións que tan só fan impresión por pantalla de cadeas de texto, pero hai moitos máis.

Definindo a función

```
index.php  x

function getSum($num1, $num2) {
    return $num1 + $num2;
}

Devolve o valor vai a continuación
(o resultado da función)
```

Valor a devolver

Chamando á función

```
index.php  x

$sum = getSum(1, 3);
echo $sum; // Saída : 4

Como se devolve o valor

function getSum($num1, $num2) {
    return $num1 + $num2;
}
```

4 será o valor a devolver



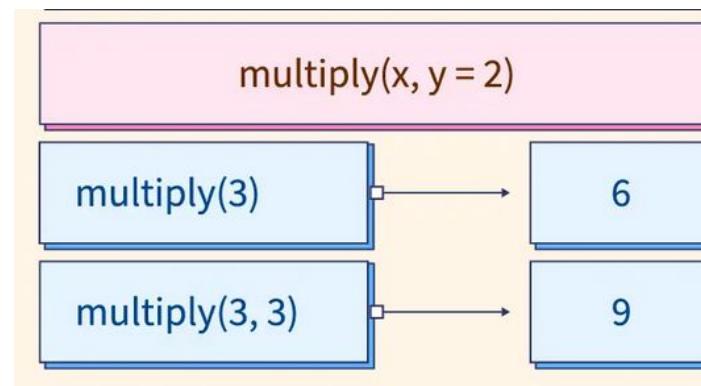
2.8 Funcións

Valores por defecto para os argumentos

- Ao definir unha función pódese indicar valores por defecto para algún(s) dos argumentos deste xeito non fai falta indicalos ao chamala. Esta forma axuda a non ter que recorrer sempre a variables globais.
- Pode haber valores por defecto para varios dos argumentos, pero sempre deben ir indicados á dereita dos que non o teñan.

```
<?php
function preco_con_ive($prezo, $ive=0.21) {
    return $prezo * (1 + $ive);
}

$prezoProduto = 10.95;
$prezoTotal = preco_con_ive($prezoProduto);
echo "<br><br>O prezo con IVE é " . $prezoTotal;
?>
```





2.8 Funcións

Parámetros por valor ou por referencia

- En todos os exemplos anteriores os argumentos pasábanse ás funcións **por valor**. Isto significa que calquera cambio que se faga dos mesmos dentro da función só será interno á mesma, ou seja local. Se queremos que a función modifique o valor dunha variable, teríamos que pasarlla **por referencia**, engadindo o símbolo & antes o argumento da función.



```
<?php
```

```
function incrementa(&$var)
{
    $var++;
}

$valor = 5;
incrementa($valor);
echo '$valor = ' . $valor . '<br>';
// $valor ten o valor 6 fora da función
?>
```

pass by reference

cup =

fillCup()

pass by value

cup =

fillCup()



pass by reference

cup =

fillCup()

pass by value

cup =

fillCup()



2.8 Funcións

Parámetros variables (I)

Podemos ter funcións nas que na declaración non indiquemos a cantidade de datos de entrada.

- `$arrayArgs = func_get_args();` → Obtén un array cos parámetros.
- `$cantidad = func_num_args();` → Obtén a cantidade de parámetros recibidos.
- `$valor = func_get_arg(numArgumento);` → Obtén o parámetro que ocupa a posición `numArgumento`.



```
<?php
function sumaParametros() {
    if (func_num_args() == 0) {
        return false;
    } else {
        $suma = 0;

        for ($i = 0; $i < func_num_args();
        $i++) {
            $suma += func_get_arg($i);
        }

        return $suma;
    }
}

echo sumaParametros(1, 5, 9); // 15
?>
```



2.8 Funcións

Parámetros variables (II)

Algo más sinxelo de lembrar é utilizando o operador **... (variadics)** o cal empaqueta os parámetros como un array (dende PHP 5.6).

```
<?php
function sumaParametros(...$numeros) {
    if (count($numeros) == 0) {
        return false;
    } else {
        $suma = 0;

        foreach ($numeros as $num) {
            $suma += $num;
        }

        return $suma;
    }
}

echo sumaParametros(10, 5, 15); // 30
?>
```



2.8 Funcións

Declaracións de tipo

- As declaracións de tipo permiten ás funcións requisitar que os parámetros sexan de certo tipo durante a chamada, así coma o valor que devolven. Se o valor non é do tipo indicado amosará un erro ou excepción. Pode antepoñerse o nome do tipo ao nome do parámetro, dentro dos seguintes tipos: `bool`, `int`, `float`, `string`, `array`, `object`, `self`, etc.

```
● ● ●  
<?php  
  
function concatena(string $a, string $b) {  
    return $a . $b;  
}  
  
echo "<br>Resultado: " . concatena("Ola!", " que pacha?");  
  
function suma(float $a, float $b) {  
    return $a + $b;  
}  
  
$s1 = (float) 1.1; // Teñen que ser float, non double  
$s1 = (float) 2.2;  
echo "<br>Resultado da suma = " . suma($s1, $s2);  
  
?>
```

Nota: Para que funcionen cómpre configurar o PHP para que permita as declaracións de tipo. **Ao comezo de cada script** habería que engadir:



```
<?php declare(strict_types=1); ?>
```

Nota: Se non devolve nada non hai por que indicalo, pero pode forzarse co tipo `void`.



```
function saudar() : void {  
    echo "Só devolve texto";  
}  
  
saudar();
```



2.8 Funcións

Declaración de retorno anulable

- A partir de PHP 7.1.0, os valores de retorno poden ser marcados como nulos antepoñendo o prefixo do tipo coa sinatura dunha interrogación (?). Isto significa que **a función devolve o tipo especificado ou null**.
- Pode facerse **o mesmo cos parámetros**, de maneira que poden coller o valor do tipo ou null.



```
<?php
function get_item(): ?string {
    if (isset($_GET['item'])) {
        return $_GET['item'];
    } else {
        return null;
    }
?>
```



```
<?php
// Declaración de función que devuelve un enteiro ou null
function obtenerNumero(?int $valor): ?int {
    return $valor;
}

// Chamada á función con valor non nulo
$resultado1 = obtenerNumero(42);
echo "Resultado 1: " . $resultado1 . "<br>"; // Saída: Resultado 1: 42

// Chamada á función con valor nulo
$resultado2 = obtenerNumero(null);
echo "Resultado 2: " . $resultado2 . "<br>"; // Saída: Resultado 2:

?>
```



2.9 Ámbitos de uso das variables

- En PHP pódense emplegar variables en calquera punto dun programa ou script. Se esa variable aínda non existe, é dicir, non se usou con anterioridade na primeira ocasión en que se fai se reserva espazo en memoria para ela. Tamén nese momento, dependendo de onde apareza, vai implícito **en que partes se poderá utilizar**, é o que se denomina **visibilidade ou ámbito** da variable.
- Se a variable aparece por primeira vez dentro dunha función, dise que é **local á función**. Se aparece tamén fóra da función, será unha variable diferente. Por exemplo:

```
// Exemplo #1 Visibilidade de variables (I)

<?php

$a = 1;

function parvada( )
{
    $b = $a;
    // Dentro da función non se accede á variable $a co valor 1,
    esta $a é nova.
    return $b;
    //Como $a non estaba asignada, $b queda con valor null.
}
?>
```



2.9 Ámbitos de uso das variables

- Para empregala dentro da función habería que declarala como **global** dentro da mesma:

```
// Exemplo #2 Visibilidade de variables (II)

<?php
$a = 1;

function parvada()
{
    global $a;
    $b = $a;
    return $b;
    //Como $a neste caso coincide, $b queda con valor 1.
}
?>
```



2.9 Ámbitos de uso das variables

- As variables locais a unha función desaparecen cando se sae da mesma, e o seu valor pérdeuse. Se se quixera manter o valor dunha variable local entre distintas chamadas a unha función, áinda que o máis elegante sería poñela fóra, pódese declarar como **estática**, e deste xeito manterase nesa función.

```
// Exemplo #3 Visibilidade de variables (III)

<?php
function contador()
{
    static $a = 0;
    $a++;
    // Cada vez que se chame a contador, $a incrementarase nunha
    // unidade
}
?>
```

Nota: As variables estáticas deben inicializarse na mesma sentenza na que se declaran como estáticas, pero só se inicializan a primeira vez que se chama a función.



2.9 Ámbitos de uso das variables

- En ocasións é necesario que unha función modifique ou **devolva máis dun valor**, pero só devolve un. ¿Como modificar máis dunha variable, ou devolver varios valores logo?
- Opcións** → Podemos facer que devolva un array, definir varias variables globais ou pasarlle varios argumentos por referencia. ¿Que método é mellor?



```
function obterInformacion() {  
    $informacion = array(  
        'nome' => 'Breogán',  
        'idade' => 30,  
        'cidade' => 'Lugo'  
    );  
    return $informacion;  
}  
  
$datos = obterInformacion();  
echo "Nome: " . $datos['nome'] . "<br>";  
echo "Idade: " . $datos['idade'] . "<br>";  
echo "Cidade: " . $datos['cidade'] . "<br>";
```



```
function obterValores(&$valor1, &$valor2) {  
    $valor1 = 'novo valor 1';  
    $valor2 = 'novo valor 2';  
}  
  
$meuValor1 = 'valor orixinal 1';  
$meuValor2 = 'valor orixinal 2';  
  
obterValores($meuValor1, $meuValor2);  
  
echo "Valor 1: " . $meuValor1 . "<br>";  
// Amosará "novo valor 1"  
echo "Valor 2: " . $meuValor2 . "<br>";  
// Amosará "novo valor 2"
```



```
$nomeUsuarioGlobal = "";  
$idadeUsuarioGlobal = 0;  
  
function encherInformacionUsuarioGlobal() {  
    global $nomeUsuarioGlobal, $idadeUsuarioGlobal;  
    $nomeUsuarioGlobal = "Breogán";  
    $idadeUsuarioGlobal = 30;  
}  
  
// Chamar a función que enche as variables globais  
encherInformacionUsuarioGlobal();  
  
// Acceder ás variables globais fóra da función  
$nomeUsuario = $nomeUsuarioGlobal;  
$idadeUsuario = $idadeUsuarioGlobal;  
  
echo "Nome do usuario: " . $nomeUsuario . "<br>";  
echo "Idade do usuario: " . $idadeUsuario . " anos<br>";
```



Documentación complementaria

- [Manual oficial de PHP.](#)
- [Wiki CIFP Rodolfo Ucha: Sintaxe básica de PHP.](#)
- [Comentarios en PHP.](#)
- [Operadores disponibles en PHP.](#)
- [Paso de variables por referencia.](#)
- [Ámbito das variables.](#)
- [Recomendacóns de estándares para PHP.](#)





Fin!

¿Algunha pregunta?

Lembra que tamén podes preguntar:

- ◉ Nos foros da aula virtual.
- ◉ Pola mensaxería da aula virtual.