Índice

Operaciones aritmética

Números

Los números decimales se escriben con punto (.), no con coma (,).

```
<?php
  $numero = 4.67;
?>
```

Un aspecto importante a recordar es que no es necesario que un número sea de tipo int si no tiene una parte decimal. Por ejemplo, 16 * 2.5 es exactamente 40, pero el tipo de este resultado aún será float. Cuando estás multiplicando números, el resultado final será de tipo flotante si al menos uno de los operandos era flotante. No importa si el número final tiene una parte decimal o no.

```
<?php
  $numero = 16 * 2.5;
  echo "<p>".gettype($numero)."</p>"; // muestra "double"
?>
```

```
<p>10</p>
```

Operadores aritméticos

Los operadores aritméticos básicos son los siguientes:

Ejemplo	Nombre	Resultado
-$a	Negación	El opuesto de $a.
$a + $b	Suma	Suma de $a y $b.
$a - $b	Resta	Diferencia entre $a y $b.
$a * $b	Multiplicación	Producto de $a y $b.
$a / $b	División	Cociente de $a y $b.
$a % $b	Módulo	Resto de $a dividido por $b.

Operadores de incremento y decremento

Ejemplo	Nombre	Efecto
++$a	Pre-incremento	Incrementa $a en uno, y luego devuelve $a.
$a++	Post-incremento	Devuelve $a, y luego incrementa $a en uno.
--$a	Pre-decremento	Decrementa $a en uno, luego devuelve $a.
$a--	Post-decremento	Devuelve $a, luego decrementa $a en uno.

La diferencia entre el pre-incremento y el post-incremento es que en el primer caso primero se incrementa la variable y después se utiliza y en el segundo primero se utiliza y después se incrementa.

```
<?php
  $valor = 9;
  echo "<p>" . ++$valor . "</p>";
?>
```

```
<p>10</p>
```

```
<?php
  $valor = 9;
  echo "<p>" . $valor++ . "</p>";
?>
```

```
<p>9</p>
```

El operador de incremento funciona también con caracteres, teniendo en cuenta que al incrementar el carácter 'z' se obtiene la cadena 'aa'. El operador de decremento no funciona con caracteres.

```
<?php
  $valor = "b";
  echo "<p>" . ++$valor . "</p>";
?>
```

```
<p>c</p>
```

```
<?php
  $valor = "z";
```

```
  echo "<p>" . ++$valor . "</p>";
?>
```

```
<p>aa</p>
```

Resto de una división

El operador % calcula el resto de una división entera

```
<?php
 echo "<p>El resto de 17 dividido entre 3 es " . 17 % 3 . "</p>";
?>
```

```
<p>El resto de 17 dividido entre 3 es 2</p>
```

La función **fmod** calcula el resto de una división con números decimales

```
<?php
 echo "<p>El resto de 17 dividido entre 3.1 es " . fmod(17, 3.1)
."</p>";
?>
```

```
<p>El resto de 17 dividido entre 3.1 es 1.5</p>
```

Paréntesis

Los operadores aritméticos tienen el mismo orden de precedencia que en Matemáticas. Concretamente, el orden de precedencia de los operadores comentados anteriormente es, de mayor a menor, el siguiente (los operadores indicados en el mismo grupo se efectúan en el orden en que aparecen en la expresión):

- ++ (incremento) -- (decremento)
- * (multiplicación) / (división) % (resto)
- + (suma) - (resta)

Los paréntesis permiten agrupar operaciones de manera que las operaciones entre paréntesis se realicen antes que las operaciones fuera de los paréntesis. Se aconseja no utilizar paréntesis cuando las operaciones den el mismo resultado con o sin paréntesis.

Operadores combinados

Los operadores combinados permiten simplificar algunas expresiones de asignación:

Ejemplo	Nombre	Equivale a
$a += $b	Suma	$a = $a + $b
$a -= $b	Resta	$a = $a - $b
$a *= $b	Multiplicación	$a = $a * $b
$a /= $b	División	$a = $a / $b
$a %= $b	Módulo	$a = $a % $b

Redondear un número

La función **round(x)** redondea el número x al entero más próximo.

```
<?php
  echo "<p>2.6 se redondea con round() a " . round(2.6) . "</p>";
  echo "<p>2.2 se redondea con round() a " . round(2.2) . "</p>";
  echo "<p>-2.6 se redondea con round() a " . round(-2.6) .
"</p>";
  echo "<p>-2.2 se redondea con round() a " . round(-2.2) .
"</p>";
?>
```

```
<p>2.6 se redondea con round() a 3</p>
<p>2.2 se redondea con round() a 2</p>
<p>-2.6 se redondea con round() a -3</p>
<p>-2.2 se redondea con round() a -2</p>
```

La función **round()** se puede utilizar para verificar si un número es un número entero (positivo o negativo), comprobando si un número coincide con su valor redondeado.

```
<?php
  $numero = 4.3;
  if ($numero == round($numero)) {
    echo "<p>$numero es un número entero</p>";
  } else {
    echp "<p>$numero no es un número entero</p>";
  }

  $numero = -6;
  if ($numero == round($numero)) {
```

```
    echo "<p>$numero es un número entero</p>";
  } else {
    echo "<p>$numero no es un número entero</p>";
  }
?>
```

```
<p>4.3 no es un número entero</p>
<p>-6 es un número entero</p>
```

La función **round(x,n)** redondea x con n decimales (si n es negativo redondea a decenas, centenas, etc.).

```
<?php
  echo "<p>2.6574 se redondea con round() con dos decimales a " .
round(2.6574, 2) . "</p>";
  echo "<p>3141592 redondeado con round() con centenas es " .
round(3141592, -2) . "</p>";
?>
```

```
<p>2.6574 se redondea con round() con dos decimales a 2.66</p>
<p>3141592 redondeado con round() con centenas es 3141600</p>
```

La función **floor(x)** redondea el número x al entero inferior (es decir, devuelve la parte entera).

```
<?php
  echo "<p>2.6 se redondea con floor() a " . floor(2.6) . "</p>";
  echo "<p>2.2 se redondea con floor() a " . floor(2.2) . "</p>";
  echo "<p>-2.6 se redondea con floor() a " . floor(-2.6) .
"</p>";
  echo "<p>-2.2 se redondea con floor() a " . floor(-2.2) .
"</p>";
?>
```

```
<p>2.6 se redondea con floor() a 2</p>
<p>2.2 se redondea con floor() a 2</p>
<p>-2.6 se redondea con floor() a -3</p>
<p>-2.2 se redondea con floor() a -3</p>
```

La función **ceil(x)** redondea el número x al entero superior.

```
<?php
  echo "<p>2.6 se redondea con ceil() a " . ceil(2.6) . "</p>";
  echo "<p>2.2 se redondea con ceil() a " . ceil(2.2) . "</p>";
```

```
  echo "<p>-2.6 se redondea con ceil() a " . ceil(-2.6) . "</p>";
  echo "<p>-2.2 se redondea con ceil() a " . ceil(-2.2) . "</p>";
?>
```

```
<p>2.6 se redondea con ceil() a 3</p>
<p>2.2 se redondea con ceil() a 3</p>
<p>-2.6 se redondea con ceil() a -2</p>
<p>-2.2 se redondea con ceil() a -2</p>
```

Potencias

La función **pow(x, y)** calcula x elevado a y.

En PHP 5.6 se introdujo el operador **, equivalente a la función **pow()**.

```
<?php
  echo "<p>2<sup>3</sup> = " . pow(2, 3) . "</p>";
?>
```

```
<p>2³ = 8</p>
```

Máximo y mínimo

Las funciones **max()** y **min()** devuelven el máximo y el mínimo, respectivamente, de una lista o matriz de valores.

```
<?php
  echo "<p>El máximo es " . max(20, 40, 25.1, 14.7) . "</p>";
?>
```

```
<p>El máximo es 40</p>
```

```
<?php
  echo "<p>El mínimo es " . min(20, 40, 25.1, 14.7) . "</p>";
?>
```

```
<p>El mínimo es 14.7</p>
```

```
<?php
  $datos = [20, 40, 25.1, 14.7];
  echo "<p>El mínimo es " . min($datos) . "</p>";
?>
```

```
<p>El mínimo es 14.7</p>
```

Formatear un número

Para escribir un número con los símbolos de separación de decimales y de miles españoles, es decir, una coma (,) para separar la parte entera de la decimal y un punto (.) para separar las cifras de la parte entera en grupos de tres, se puede utilizar la función **number_format()**.

```
<?php
  print "<p>Format: " . number_format(1300, 5) . "</p>";
?>
```

```
<p>>Format:  1,300.00000</p>
```

```
<?php
  print "<p>Format: " . number_format(123456.789, 2) . "</p>";
?>
```

```
<p>Format: 123,456.79</p>
```

```
<?php
   print "<p>Format: " . number_format(123456789123, 0, ",", ".")
. "</p>";
?>
```

```
<p>123.456.789.123</p>
```

```
<?php
   print "<p>Format: " . number_format(123456789123, 2, ",", ".")
. "</p>";
?>
```

```
<p>123.456.789.123.456,78</p>
```

La función requiere dos o cuatro argumentos:

- El primer argumento es el número a formatear.
- El segundo argumento es el número de decimales a mostrar (el número se redondea o trunca dependiendo de la longitud del número).
- El tercer argumento es el carácter a utilizar como separador de la parte entera de la decimal.
- El cuarto argumento es el carácter a utilizar como separador de miles.

La función devuelve el número formateado. Si sólo se utilizan dos argumentos, se utiliza el punto como separador de parte entera y decimal y la coma como separador de miles (notación inglesa).

Números aleatorios

Para obtener un número entero aleatorio entre dos valores determinados, se pueden utilizar la función **rand()** o la función **mt_rand()**. Ambas funciones requieren dos argumentos:

- El primer argumento es el valor mínimo que se quiere obtener
- El segundo argumento es el valor máximo que se quiere obtener.

```
<?php
  echo "<p>" . mt_rand(1, 6) . "</p>";
  echo "<p>" . mt_rand(1, 6) . "</p>";
  echo "<p>" . mt_rand(1, 6) . "</p>";
?>
```

```
<p>6</p>
<p>2</p>
<p>4</p>
```

```
<?php
  echo "<p>" . rand(-10, 10) . "</p>";
  echo "<p>" . rand(-10, 10) . "</p>";
  echo "<p>" . rand(-10, 10) . "</p>";
?>
```

```
<p>-8</p>
<p>-10</p>
<p>3</p>
```

Si se llama a las funciones sin argumentos, estas devuelven un número entero aleatorio entre 0 y un valor máximo que es el que devuelve la función **getrandmax()** o **mt_getrandmax()** (este valor depende del sistema operativo).

```
<?php
  echo "<p>" . mt_getrandmax() . "</p>";
  echo "<p>" . mt_rand() . "</p>";
  echo "<p>" . mt_rand() . "</p>";
?>
```

```
<p>2147483647</p>
<p>124189893</p>
```

```
<p>1144610897</p>
```

```
<?php
  echo "<p>" . getrandmax() . "</p>";
  echo "<p>" . rand() . "</p>";
  echo "<p>" . rand() . "</p>";
?>
```

```
<p>32767</p>
<p>18805</p>
<p>20361</p>
```

Nota: En algunas plataformas (como en Windows), **getrandmax()** sólo alcanza hasta 32767. En caso de necesitar un valor mayor de 32767 se debería emplear **mt_rand()** en lugar de **rand()**.

Operaciones lógicas

Las operaciones lógicas son expresiones matemáticas cuyo resultado es un valor booleano (verdadero o falso, en PHP, **true** o **false**). Estas expresiones se utilizan principalmente en las estructuras de control.

Comparaciones

Las comparaciones permiten comparar variables o expresiones entre sí o con valores concretos. El resultado de la comparación es un valor booleano (**true** o **false**).

Ejemplo	**Nombre**	**Resultado**
expresión_1 == expresión_2	Igual	**true** si expresión_1 es igual a expresión_2.
expresión_1 === expresión_2	Idéntico	**true** si expresión_1 es igual a expresión_2, y son del mismo tipo. (a partir de PHP 4)
expresión_1 != expresión_2	Diferente	
expresión_1 <> expresión_2		
expresión_1 !== expresión_2	No idénticos	**true** si expresión_1 no es igual a expresión_2, o si no son del mismo tipo. (a partir de PHP 4)

expresión_1 < expresión_2	Menor que	**true** si expresión_1 es estrictamente menor que expresión_2.
expresión_1 > expresión_2	Mayor que	**true** si expresión_1 es estrictamente mayor que expresión_2.
expresión_1 <= expresión_2	Menor o igual que	**true** si expresión_1 es menor o igual que expresión_2.
expresión_1 >= expresión_2	Mayor o igual que	true si expresión_1 es mayor o igual que expresión_2.

```
<?php
  $nombre = "Pepe";

  if ($nombre == "Juan") {
    echo "<p>Tu nombre es Juan.</p>";
  }
  if ($nombre != "Juan") {
    echo "<p>Tu nombre no es Juan.</p>";
  }
?>
```

```
<p>Tu nombre no es Juan.</p>
```

```
<?php
  $nombrePadre = "Pepe";
  $nombreHijo = "Pepe";

  if ($nombrePadre == $nombreHijo) {
    echo "<p>El hijo se llama como el padre.</p>";
  }
  if ($nombrePadre != $nombreHijo) {
    echo "<p>El hijo no se llama como el padre.</p>";
  }
?>
```

```
<p>El hijo se llama como el padre.</p>
```

Así como en las asignaciones la variable se escribe siempre a la izquierda de la igualdad, en una comparación las variables pueden aparecer a la derecha de la comparación:.

```
<?php
```

```
  $nombre = "Pepe";

  if ("Juan" == $nombre) {
    echo "<p>Tu nombre es Juan.</p>";
  }
  if ("Juan" != $nombre) {
    echo "<p>Tu nombre no es Juan.</p>";
  }
?>
```

```
<p>Tu nombre no es Juan.</p>
```

Un error típico cuando se empieza a programar es confundir el operador comparación (==) con el operador de asignación (=), lo que produce resultados inesperados, como en el ejemplo siguiente:

```
<?php
  $nombre = "Pepe";

  if ($nombre = "Juan") {
    echo "<p>Tu nombre es Juan.</p>";
  }
  if ( $nombre != "Juan") {
    echo "<p>Tu nombre no es Juan.</p>";
  }
?>
```

```
<p>Tu nombre es Juan.</p>
```

Al escribir = en vez de ==, PHP no compara el valor de la variable $nombre con la cadena "Juan", sino que cambia el valor de $nombre a "Juan". Como la asignación no da error, PHP le asigna el valor booleano **true**, y escribe que el nombre es Juan.
Al no mostrar PHP ningún aviso o mensaje de error, este error puede ser difícil de localizar en programas largos o complejos.
En principio no se pueden concatenar comparaciones:

```
<?php
  $entero = 6;
  $cadena = "6";
  $decimal = 6.0;

  if ($entero == $cadena == $decimal) {
    echo "<p>$entero, $cadena y $decimal son iguales.</p>";
  } else {
```

```
    echo "<p>$entero, $cadena y $decimal son distintos.</p>";
  }
?>
```

```
Parse error: syntax error, unexpected '==' (T_IS_EQUAL) in ejemplo.php on line 6
```

En estos casos habría que concatenar comparaciones mediante operadores lógicos, como se explica más adelante.

```
<?php
  $entero = 6;
  $cadena = "6";
  $decimal = 6.0;

  if ($entero == $cadena && $entero == $decimal) {
    echo "<p>$entero, $cadena y $decimal son iguales.</p>";
  } else {
    echo "<p>$entero, $cadena y $decimal son distintos.</p>";
  }
?>
```

```
<p>6, 6 y 6 son iguales.</p>
```

Hay que tener cuidado al comparar expresiones de tipos diferentes, ya que se pueden producir resultados inesperados.
PHP permite comparar expresiones de tipos distintos, y la comparación es correcta siempre que sea posible interpretar un dato de un tipo como un dato de otro tipo.

- Podemos comparar números enteros y decimales:

```
<?php
  $entero = 5;
  $decimal = 5.0;

  if ($entero == $decimal) {
    echo "<p>$entero y $decimal son iguales.</p>";
  } else {
    echo "<p>$entero y $decimal son distintos.</p>";
  }
?>
```

```
<p>5 y 5 son iguales.</p>
```

- Un número escrito como número (sin comillas) o como texto (con comillas) son iguales para PHP ...:

```php
<?php
  $numero = 5;
  $cadena = "5";

  if ($numero == $cadena) {
    echo "<p>$numero y $cadena son iguales.</p>";
  } else {
    echo "<p>$numero y $cadena son distintos.</p>";
  }
?>
```

```
<p>5 y 5 son iguales.</p>
```

- ... pero PHP no efectúa cálculos en las cadenas:

```php
<?php
  $numero = 4 + 1;
  $cadena = "4+1";

  if ($numero == $cadena) {
    echo "<p>$numero y $cadena son iguales.</p>";
  } else {
    echo "<p>$numero y $cadena son distintos.</p>";
  }
?>
```

```
<p>5 y 4+1 son distintos.</p>
```

- Además, hay situaciones especiales. Por ejemplo, cuando una cadena empieza por un número, PHP interpreta la cadena como número y descarta el resto:

```php
<?php
  $numero = 5;
  $cadena = "5abc def";

  if ($numero == $cadena) {
    echo "<p>$numero y $cadena son iguales.</p>";
  } else {
    echo "<p>$numero y $cadena son distintos.</p>";
  }
?>
```

```
<p>5 y 5abc def son iguales.</p>
```

```php
<?php
  $numero = 5;
  $cadena = "5+1";

  if ($numero == $cadena) {
    echo "<p>$numero y $cadena son iguales.</p>";
  } else {
    echo "<p>$numero y $cadena son distintos.</p>";
  }
?>
```

```
<p>5 y 5+1 son iguales.</p>
```

Para evitar problemas con las conversiones automáticas de tipo se pueden utilizar los operadores **===** o **!==**. La diferencia entre estos operadores y los operadores **==** y **!=** es que aquellos comprueban además si los tipos son iguales.

```php
<?php
  $numero = 5;
  $texto = "5";

  if ($numero == $texto) {
    echo "<p>Las variables son iguales.</p>";
  } else {
    echo "<p>Las variables no son iguales.</p>";
  }
?>
```

```
<p>Las variables son iguales.</p>
```

```php
<?php
  $numero = 5;
  $texto = "5";

  if ($numero === $texto) {
    echo "<p>Las variables son idénticas.</p>";
  } else {
    echo "<p>Las variables no son idénticas.</p>";
  }
?>
```

```
<p>Las variables no son idénticas.</p>
```

Operadores lógicos

Los operadores lógicos permiten combinar expresiones simples en expresiones más complejas.

Ejemplo	Nombre	Resultado
expresión_1 && expresión_2	Y	**true** si los dos, expresión_1 y expresión_2, son true.
expresión_1 and expresión_2		
expresión_1 || expresión_2	O	**true** si uno de los dos, expresión_1 o expresión_2, es true.
expresión_1 or expresión_2		
expresión_1 xor expresión_2	O exclusivo (Xor)	**true** si sólo uno de los dos, expresión_1 o expresión_2, es true, pero no ambos.
!expresión	Negación	**true** si expresión no es true.

Al escribir expresiones en las que se combinan varias comparaciones mediante operadores lógicos es conveniente utilizar paréntesis, aunque en muchos casos no sean necesarios porque las comparaciones tienen precedencia sobre los operadores lógicos.

Diferencia entre and y && y entre or y ||

Los operadores **and** y **&&** y los operadores **or** y **||** no son completamente equivalentes, ya que no tienen la misma preferencia. Concretamente, **&&** y **||** tienen mayor prioridad que **and** y **or**. Como además el operador de asignación **=** tiene una prioridad intermedia, se pueden producir situaciones inesperadas, como muestran los siguientes ejemplos.
El ejemplo siguiente muestra el resultado esperado:

```
<?php
  $var1 = true;
  $var2 = false;
  $union = $var1 && $var2;

  if ($union) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>falso</p>
```

La variable $union sólo tomaría el valor **true** si tanto $var1 como $var2 fueran **true**, pero como $var2 es **false**, $union toma el valor **false**.
Sin embargo si se utiliza el operador **and** en vez de **&&,** el resultado no es el esperado:

```
<?php
  $var1 = true;
  $var2 = false;
  $union = $var1 and $var2;

  if ($union) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>verdadero</p>
```

¿Por qué se produce ese resultado? Porque el operador de asignación = tiene preferencia sobre el operador **and**. Eso quiere decir que PHP realiza antes la asignación que la operación lógica, es decir, como si la expresión estuviese escrita así:

```
<?php
  $var1 = true;
  $var2 = false;
  ($union = $var1) and $var2;
  if ($union) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>verdadero</p>
```

En esa expresión, la variable $union almacena el valor de la variable $var1 (**true**), por lo que $todo toma el valor **true**. La operación lógica **and** no modifica el valor de $union.
Si se quiere obtener el mismo resultado con **and** que con **&&**, se deben utilizar paréntesis, para forzar que la operación lógica **and** se realice antes de la asignación:

```
<?php
```

```
  $var1 = true;
  $var2 = false;
  $union = ($var1 and $var2);

  if ($union) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>falso</p>
```

En el caso de los operadores **or** y **||** ocurre lo mismo:

```
<?php
  $var1 = true;
  $var2 = false;
  $otro = $var2 || $var1;

  if ($otro) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>verdadero</p>
```

La variable $otro tomará el valor **true** si bien $var1 o bien $var2 son **true**, como $var1 es **true**, $otro toma el valor **true**.
Sin embargo si se utiliza el operador **or** en vez de **||**, el resultado no es el esperado:

```
<?php
  $var1 = true;
  $var2 = false;
  $otro = $var2 or $var1;

  if ($otro) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>falso</p>
```

¿Por qué se produce ese resultado? Porque el operador de asignación = tiene precedencia sobre el operador or. Eso quiere decir que PHP realiza antes la asignación que la operación. La variable $otro almacena el valor de la variable $var2 (false).
Si se quiere obtener el mismo resultado con **or** que con **||**, se deben utilizar paréntesis, para forzar que la operación lógica or se realice antes de la asignación:

```
<?php
  $var1 = true;
  $var2 = false;
  $todo = ($var2 or $var1);

  if ($todo) {
    echo "<p>verdadero</p>";
  } else {
    echo "<p>falso</p>";
  }
?>
```

```
<p>verdadero</p>
```