

MÓDULO 2ºDAW: **Desarrollo Web en entorno Servidor.**

Autor: Jorge López.



Tema 01: Sintaxis básica en PHP - Parte 2ª

06 Operadores

- Operador de asignación
- Operador Unario
- Operadores Aritméticos
- Operadores de Comparación
- Operadores Lógicos
- Operador Ternario
- Operadores Bit a Bit
- Operadores de asignación Combinados
- Precedencia de Operadores

07 Estructuras de Control

08 Estructuras de Elección

- Estructura IF-ELSE
- Estructura ELSEIF
- Estructura SWITCH

09 Estructuras de Bucle

- WHILE
- DO-WHILE
- FOR
- BREAK Y CONTINUÉ

06 Operadores.

Vamos a ver un resumen de los operadores más utilizados en PHP:

Operador de asignación

El más básico es el símbolo de asignación (=), utilizado para dar valores a las variables que usamos en nuestro código.

```
<?php
$variable = 34;
$variable2 = "Asignación de valores";
?>
```

Las variables que están a la izquierda del operador toman el valor que se encuentra en la expresión de la derecha.

Advertencia: no hay que confundir el operador de asignación (=) con el operador de comparación (==), sobre todo en bucles de control, donde sería difícil encontrar algún fallo.

Operador Unario

El signo menos (-) se utiliza delante de un número o variable numérica. Este operador tiene la propiedad de hacer a los números, negativos o positivos, dependiendo del signo actual.

```
<?php
$entero = 23;
$entero_negativo = -$entero; // El valor es ahora -23
$entero2 = -$entero_negativo; // El valor cambia ahora a 23
?>
```

Operadores Aritméticos

Este tipo de operadores forman parte de la aritmética básica. Nos resultará familiar porque son símbolos muy utilizados en el aprendizaje de las matemáticas.

| Operadores Aritméticos | | |
|------------------------|----------------|---------------------------------------|
| Ejemplo | Nombre | Resultado |
| \$a + \$b | Suma | Suma las dos variables |
| \$a - \$b | Resta | Resta las dos variables |
| \$a * \$b | Multiplicación | Producto de las variables |
| \$a / \$b | División | Cociente entre las dos variables |
| \$a % \$b | Módulo | Resto de la división de \$a entre \$b |

Operadores de Comparación

Vamos a ver un resumen de este tipo de operadores, el resultado de utilizar un operador de comparación siempre es TRUE o FALSE.

| Operadores de Comparación | | |
|--|---------------|---|
| Ejemplo | Nombre | Resultado |
| <code>\$a == \$b</code> | Igualdad | Devuelve true si \$a y \$b son iguales |
| <code>\$a === \$b</code> | Identidad | Devuelve true si son iguales y del mismo tipo |
| <code>\$a != \$b</code> <code>\$a <> \$b</code> | Distinto | Devuelve true si \$a y \$b son distintas |
| <code>\$a < \$b</code> | Menor que | Devuelve true si \$a es menor que \$b |
| <code>\$a > \$b</code> | Mayor que | Devuelve true si \$a es mayor que \$b |
| <code>\$a <= \$b</code> | Menor o igual | Devuelve true si \$a es menor o igual que \$b |
| <code>\$a >= \$b</code> | Mayor o igual | Devuelve true si \$a es mayor o igual que \$b |

Operadores Lógicos

Durante el desarrollo de un proyecto, podemos encontrarnos con situaciones en las que necesitemos hacer varias comparaciones seguidas para que se cumpla una determinada condición. PHP 5 permite unir todas las comparaciones en una mediante el uso de los operadores lógicos.

| Operadores Lógicos | | |
|---|-------------|---|
| Ejemplo | Nombre | Resultado |
| Expresion1 and Expresion2 Expresion1 && Expresion2 | Y | Si las dos expresiones son verdaderas el valor es true |
| Expresion1 or Expresion2 Expresion1 Expresion2 | O | Si una de las expresiones es verdadera el valor es true |
| Expresion1 xor Expresion2 | O Exclusivo | True si una expresión es verdadera y la otra falsa |
| ! Expresion1 | Negación | Verdadero si la expresión no es cierta |

```
<?php
$a = 30; $b = 25; $c = 10;
if ($a < ($b + $c))
{
    echo "Se cumple la condición";
}
?>
```

Los operadores que hemos visto hasta ahora son capaces de manejar un operando (Unarios) o dos operandos (binarios). El operador ternario, o de comparación, evalúa un operando y, dependiendo de si es falso o verdadero, evalúa el segundo operando o el tercero.

La expresión que se quiere evaluar se escribe delante de un símbolo **?**, después la expresión que tiene que ejecutarse si la evaluación anterior es true, seguida del símbolo **:** con la expresión que debe ejecutarse si es false.

```
<?php
    $valor = false;
    $valor == true ? $resultado = "OK" : $resultado = "FALLO";
    // Si $value es true el resultado será OK
    // Si $value es false el resultado será FALLO
    echo $resultado;
?>
```

Operadores Bit a Bit

Este tipo de operadores rara vez será necesario utilizarlos en nuestras aplicaciones PHP.

Su utilidad suele limitarse a la gestión de periféricos y algunas operaciones de cálculo de carácter muy reiterativo en la que se puede conseguir un rendimiento muy superior a los operadores tradicionales.

En el ámbito propio del PHP pueden tener algún interés a la hora de elaborar rutinas para encriptar el código fuente de algunos scripts que por su importancia pueden requerir ese tipo de protección.

Puesto que la notación binaria se escapa del enfoque de este curso te recomiendo que busques información si es que tienes curiosidad. Probablemente no utilizarás nunca este tipo de operadores, pero eso dependerá de la magnitud y tipo de los proyectos a los que te enfrentes.

Operadores de asignación Combinados

En numerosas ocasiones se nos presentan situaciones en las que una variable debe incrementar o disminuir su valor en 1.

PHP 5-PHP 7 provee operadores combinados que permiten asignar rápidamente incrementos de valor, concatenaciones de caracteres, etc.

Veamos algunos ejemplos de estos operadores:

| Operadores de asignación Combinados | | |
|-------------------------------------|---------------|---|
| Ejemplo | Nombre | Equivalencia |
| \$a++ | Incremento | \$a = \$a +1 (primero asignación y luego suma) |
| \$a-- | Resta | \$a = \$a -1 (primero asignación y luego resta) |
| ++\$a | Incremento | \$a = \$a +1 (primero suma y luego asignación) |
| --\$a | Resta | \$a = \$a -1 (primero resta y luego asignación) |
| \$a .= \$b | Concatenación | \$a = \$a . \$b |
| \$a += \$b | Suma | \$a = \$a + \$b |
| \$a -= \$b | Resta | \$a = \$a - \$b |

```
<?php
$a = 23;
$b = $a++; // $b es igual a 23 porque $a se incrementa después de la asignación
echo "la variable b es igual a: ". $b . "<BR>";
echo "la variable a es igual a: ". $a . "<BR>";
$a = 23;
$c = ++$a; // $c es igual a 24 porque $a se incrementa antes de la asignación
echo "la variable c es igual a: ".$c."<br>";
echo "la variable a es igual a: ".$a."<br>";
?>
```

Precedencia de Operadores

Vamos a echar un vistazo al código siguiente:

```
<?php
$resultado = 20 + 10 * 10;
echo $resultado;
?>
```

Aparentemente es un código muy simple, pero encierra un problema matemático. Hay que tener mucho cuidado y **tener muy claro la precedencia y orden de los operadores a la hora de ser ejecutados:**

Existen dos resultados posibles en función del operador que debe ejecutarse antes.

Si la suma se ejecuta antes, la variable **\$resultado** tendrá el valor **300** y si la multiplicación se ejecuta antes, tendremos que **\$resultado** equivale a **120**. Si comprobamos el código, el resultado que nos muestra es 120, por lo tanto el operador de multiplicación (*) es preferente con respecto al operador de suma (+).

De cara a la legibilidad del código hubiese sido mejor escribir:

```
$resultado = 20 + (10 * 10);
```

Vamos a echar un vistazo al código siguiente:

```
<?php
$resultado = 20 / 10 / 2;
echo $resultado;
?>
```

Si los operadores que aparecen son idénticos, existe un orden de ejecución que puede ser desde la izquierda a la derecha o derecha a izquierda.

El valor de \$resultado es ahora **1**, porque primero se calcula $20 / 10$ y después ese mismo resultado se divide entre 2, es decir, la asociación entre operadores de división es desde la izquierda.

De cara a la legibilidad del código hubiese sido mejor escribir:

$\$resultado = (20 / 10) / 2;$

07 Estructuras de Control.

Es difícil imaginar un programa sin estructuras de control. Éstas nos permiten elegir diferentes caminos en la ejecución del programa o repetir varias veces la ejecución de un mismo trozo de código en función de una serie de datos que evaluaremos en cada momento. En este capítulo trataremos dos tipos de estructuras de control:

○ Estructuras de elección:

Este tipo de operadores son el auténtico cerebro de cualquier aplicación que desarrollemos en PHP o en cualquier otro lenguaje de programación.

Las estructuras de elección permiten evaluar una condición o varias y elegir el camino correcto. Los operadores condicionales son la herramienta que permite tomar decisiones tales como: hacer o no hacer, y también: hacer algo bajo determinadas condiciones y otra cosa distinta en caso de que no se cumplan.

○ Estructuras de bucle:

La necesidad de repetir la ejecución de instrucciones es algo habitual en el mundo de la programación. Frente a la alternativa poco práctica de describir esas instrucciones todos los lenguajes de programación disponen de funciones que pueden ejecutar un bloque de instrucciones de forma repetitiva.

Las estructuras de bucle repiten un número determinado de veces un conjunto de instrucciones.

08 Estructuras de Elección.

Estructura IF-ELSE

La sintaxis de esta estructura es:

```
if (condición) instrucción1;  
else instrucción2;
```

Si se cumple la condición se ejecuta la instrucción que le sigue. Si quieres que se ejecuten varias instrucciones, debe utilizar el símbolo llave (**{ instrucciones}**).

```
<?php  
if (condición)  
{  
    Instruccion1;  
    instruccion2;  
    instruccion3;  
}  
?>
```

La es-

estructura **if** puede ampliarse para que se pueda elegir entre una condición verdadera o falsa:

```
<?php
    if (condición)
    {
        Instruccion1;
        instrucción2;
        instruccion3;
    }
    else
    {
        Instruccion1;
        instrucción2;
        instruccion3;
    }
?>
```

Si la condición es verdadera se ejecutan las instrucciones inmediatamente después del **if** y si la condición es falsa se ejecutan las instrucciones dispuestas después de la palabra **else**.

Vamos a ver un **ejemplo** concreto:

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
    $valor1 = 23;
    $valor2 = 27;
    if ($valor1 < $valor2)
    {
        echo "La variable valor1 es menor que valor2";
    }
    else
    {
        echo "La variable valor1 es mayor que valor2";
    }
?>
```


Estructura ELSEIF

Es muy común en programación realizar comparaciones en cascada para comprobar una serie de valores, esta estructura **elseif** nos va a permitir realizar esas comparaciones en cascada:

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
$día=3;
if ($día == 1)
{
    echo "El día es Lunes";
}
elseif ($día == 2)
{
    echo "El día es Martes";
}
elseif ($día == 3)
{
    echo "El día es Miércoles";
}
else
{ echo "Otro día"; }
?>
```

El ejemplo anterior evalúa la variable **\$día** hasta que encuentra el valor correcto y ejecuta las instrucciones asociadas a ese valor.

Estructura switch

La construcción **switch** comprueba el valor de una expresión y dependiendo del valor de dicha expresión permite elegir entre un conjunto de instrucciones. El formato es el siguiente:

```
<?php
switch (expresión)
{
    case valor1 :
        instruccion1;
        instruccion2;
        instruccion3;
        break;
    case valor2 :
        instruccion1;
        instruccion2;
        instruccion3;
        break;
    default:
        instruccion1;
        instruccion2;
        instruccion3;
}
?>
```

La expresión puede ser de cualquier tipo, siempre que devuelva un valor de tipo entero, de coma flotante o de cadena de caracteres. Una vez evaluada la expresión, se busca el valor en la instrucción case y, si coincide, se ejecutan todas las instrucciones hasta la palabra reservada **break**. Si no coincide ningún valor, se ejecutan las instrucciones por defecto.

Veamos un ejemplo completo:

Crea un fichero llamado **prueba.php** y prueba este código:

```
<?php
$día = 4 ;
switch ($día)
{
    case 1:
        echo "El día es Lunes";
        break;
    case 2:
        echo "El día es Martes";
        break;
    case 3:
        echo "El día es Miércoles";
        break;
    case 4:
        echo "El día es Jueves";
        break;
    case 5:
        echo "El día es Viernes";
        break;
    case 6:
        echo "El día es Sábado";
        break;
    case 7:
        echo "El día es Domingo" ;
        break;
    default:
        echo "El día de la semana es incorrecto";
}
```

09 Estructuras de Bucle.

Las **estructuras de bucle** repiten un número determinado de veces un conjunto de instrucciones. Vamos a ver los distintos tipos de estructuras de bucle que existen:

WHILE

El bucle **while** es el más básico de todos. La construcción básica es la siguiente:

```
<?php
while (condición)
{
    instrucción1;
    instrucción2;
    instrucción3;
}
?>
```

La condición se evalúa al principio. Si **es verdadera**, se ejecutan las instrucciones que están dentro del bucle y se vuelve a evaluar la condición. Si la condición **es falsa** no se ejecutan las instrucciones, se sale del bucle y se continúa con el desarrollo del programa.

Puesto que la condición se evalúa antes de ejecutar las instrucciones, es posible que algunos bucles utilizando **while** no se ejecuten ninguna vez.

El siguiente ejemplo muestra una instrucción **while** que no se ejecutará nunca, porque la condición siempre será falsa:

```
<?php
$variable = 100;
while ($variable<20)
{
    echo "Esta línea no se ejecuta nunca";
}
?>
```

Hay que tener cuidado en las estructuras de bucle para que estas no se ejecuten indefinidamente, existe la posibilidad de que un bucle se ejecute **infinitas veces**, si dentro de las instrucciones no existe nada que cambie la condición que se evalúa al principio:

```
<?php
$variable = 100;
while ($variable>20)
{
    echo "CUIDADO: Esta línea se ejecuta siempre, bucle infinito";
}
?>
```

DO-WHILE

Este bucle es igual que el anterior, pero la condición se evalúa al final, después de la ejecución de las instrucciones. Por lo tanto, el código que está entre las llaves **siempre** se ejecuta al menos una vez. El formato básico es el siguiente:

```
<?php
do
{
    Instrucción1;
    instrucción2;
    instrucción3;
}
while (condición);
?>
```

FOR

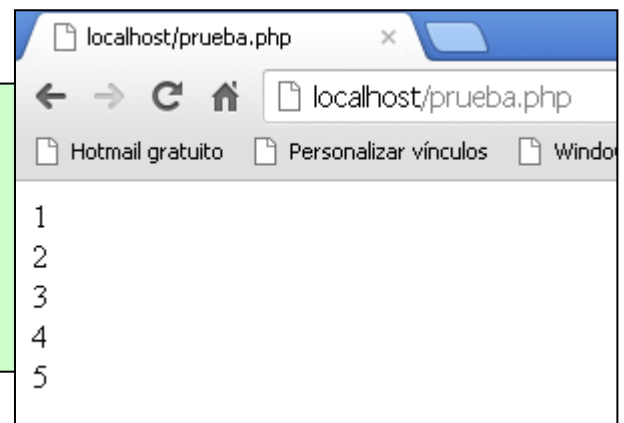
La construcción de bucle más complicada es la del for. Tiene la siguiente sintaxis:

```
<?php
for (expresión inicial; condición de fin; expresión de fin)
{
    Instrucción1;
    instrucción2;
    instrucción3;
}
?>
```

El funcionamiento es el siguiente: la expresión inicial se ejecuta una sola vez al principio del bucle. La condición de fin se evalúa cada vez que se ejecuta el bucle. Si es verdadera se continúa la ejecución y si es falsa se sale del bucle. Al final de cada interacción se ejecuta la expresión de fin.

Veamos un ejemplo:

```
<?php
for ($prueba = 1; $prueba <= 5; $prueba++)
{
    echo $prueba."<br>";
}
?>
```



BREAK Y CONTINUE

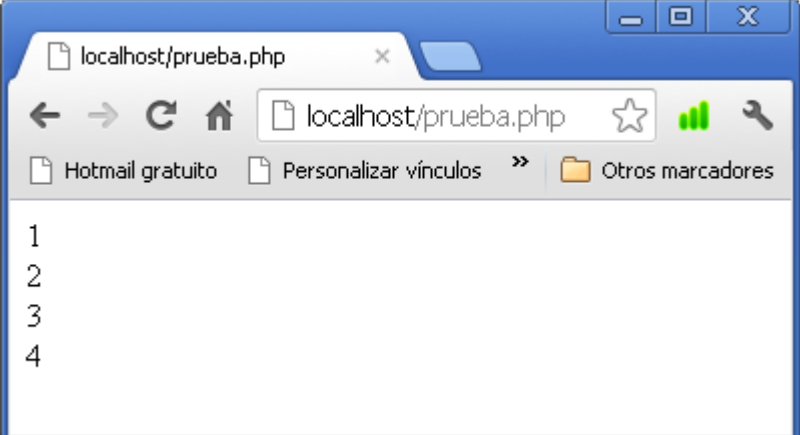
El camino ordinario y normal para salir de un bucle es que la condición se evalúe a false.

Existe una forma especial de salir de un bucle, utilizando las palabras reservadas break y continue. Su forma de actuar es la siguiente:

- break: sale del bucle actual y continúa el programa.
- continue: se usa dentro de la estructura del bucle para saltar la iteración actual del bucle y continuar con la ejecución del bucle al comienzo de la siguiente iteración.

El código siguiente muestra la forma de utilizar break:

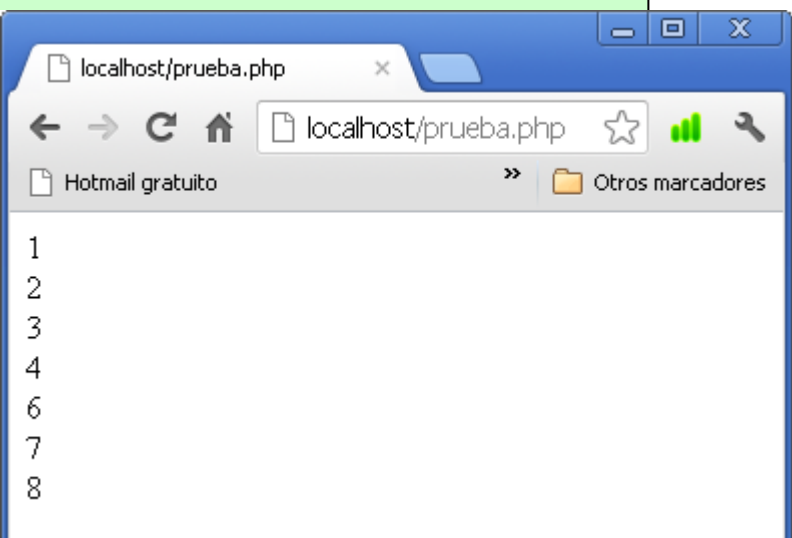
```
<?php
for ($x = 1; $x <= 8; $x++)
{
    if ($x == 5)
    {break;}
    else
    {echo "$x<br>";}
}
?>
```



La salida por pantalla es una sucesión de números del 1 al 8. Al llegar la variable \$x al número 5, la condición del if se cumple y se ejecuta la instrucción break, saliendo del bucle.

El código siguiente muestra la forma de utilizar continue:

```
<?php
for ($x = 1; $x <= 8; $x++)
{
    if ($x == 5)
    {continue;}
    else
    {echo "$x<br>";}
}
?>
```



En este caso, cuando \$x alcanza el valor 5, lo que hace la instrucción continue es evitar que se ejecuten las instrucciones contenidas en el else para esa iteración, después se continua con la ejecución del bucle. El resultado por pantalla es una sucesión de números desde el 1 hasta el 8, sin incluir el número 5.