PHP | Estructuras de control

Listaremos algunas de las estructuras de control que existen en PHP, son los building blocks para definir la lógica en tus programas, y poder crear diferentes flujos de ejecución.

Empezaremos por el más sencillo de todos.

if

Permite la ejecución condicional de fragmentos de código.

La expresión es evaluada a su valor booleano. Si la expresión se evalúa como **TRUE**, PHP ejecutará la sentencia y si se evalúa como **FALSE** la ignorará.

Ej:

```
if ($a > $b) {
    echo "a es mayor que b";
}
```

Las sentencias *if* pueden anidarse dentro de otra sentencias *if* infinitamente

else

Con frecuencia se desea ejecutar una sentencia si una determinada condición se cumple y una sentencia diferente si la condición no se cumple. Esto es para lo que sirve *else*. El *else* extiende una sentencia *if* para ejecutar una sentencia en caso que la expresión en la sentencia *if* se evalúe como **FALSE**.

Ej:

```
if ($a > $b) {
    echo "a es mayor que b";
} else {
    echo "a NO es mayor que b";
}
```

while

Los bucles *while* son el tipo más sencillo de bucle en PHP.

El significado de una sentencia *while* es simple. Le dice a PHP que ejecute las sentencias anidadas, tanto como la expresión *while* se evalúe como **TRUE**. El valor de la expresión es verificado cada vez al inicio del bucle, por lo que incluso si este valor cambia durante la ejecución de las sentencias anidadas, la ejecución no se detendrá hasta el final de la iteración (cada vez que PHP ejecuta las sentencias contenidas en el bucle es una iteración). A veces, si la expresión *while* se evalúa como **FALSE** desde el principio, las sentencias anidadas no se ejecutarán ni siquiera una vez.

Ej: mostrar los número del 1 al 10

```
$i = 1;
while ($i <= 10) {
  echo $i++; /* el valor presentado sería $i antes del incremento (post-incremento) */
}
```

for

Los bucles for son los más complejos en PHP.

La sintaxis de un bucle for es:

> for (expr1; expr2; expr3)
> sentencia

La primera expresión (expr1) es evaluada (ejecutada) una vez incondicionalmente al comienzo del bucle.

En el comienzo de cada iteración, se evalúa expr2. Si se evalúa como TRUE, el bucle continúa y se ejecutan la/sy sentencia/s anidada/s. Si se evalúa como FALSE, finaliza la ejecución del bucle.

Al final de cada iteración, se evalúa (ejecuta) expr3.

Cada una de las expresiones puede estar vacía o contener múltiples expresiones separadas por comas. En expr2, todas las expresiones separadas por una coma son evaluadas, pero el resultado se toma de la última parte. Que expr2 esté vacía significa que el bucle debería ser ejecutado indefinidamente (PHP implícitamente lo considera como TRUE). Esto puede no ser tan inútil como se pudiera pensar, ya que muchas veces se debe terminar el bucle usando una sentencia condicional break en lugar de utilizar la expresión verdadera del for.

Ej: 2 maneras diferentes de mostrar los números de 1 al 10

```
/* ejemplo 1 */
for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* ejemplo 2 */
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* ejemplo 2 */
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
```

foreach

El constructor *foreach* proporciona un modo sencillo de iterar sobre arrays. *foreach* funciona sólo sobre arrays y objetos, y emitirá un error al intentar usarlo con una variable de un tipo diferente de datos o una variable no inicializada. Existen dos sintaxis:

> foreach (expresión_array as $valor)
> sentencias
>
> foreach (expresión_array as $clave ⇒ $valor)
> sentencias

La primera forma recorre el array dado por *expresión_array*. En cada iteración, el valor del elemento actual se asigna a *$valor* y el puntero interno del array avanza una posición (así en la próxima iteración se estará observando el siguiente elemento).

La segunda forma además asigna la clave del elemento actual a la variable *$clave* en cada iteración.

Algunos ejemplos:

```
<?php
/* Ejemplo 1 de foreach: sólo el valor */
$a = [1, 2, 3, 17];

foreach ($a as $v) {
    echo "Valor actual de \$a: $v.\n";
}

/* Ejemplo 2 de foreach: clave y valor */
$a = ["uno" => 1, "dos" => 2, "tres" => 3, "diecisiete" => 17];
foreach ($a as $k => $v) {
    echo "\$a[$k] => $v.\n";
}
```

break

break finaliza la ejecución de la estructura *for*, *foreach*, *while*, *do-while* o *switch* en curso.

break acepta un argumento numérico opcional que indica de cuántas estructuras anidadas circundantes se debe salir. El valor predeterminado es *1*, es decir, solamente se sale de la estructura circundante inmediata.

Ej:

```
$arr = array('uno', 'dos', 'tres', 'cuatro', 'pare', 'cinco');
foreach ($arr as $val) {
    if ($val == 'pare') {
        break; /* Se puede también escribir 'break 1;' aquí. */
    }
    echo "$val\n";
}

/* Utilizar el argumento opcional. */
$i = 0;
while (++$i) {
    switch ($i) {
        case 5:
            echo "En 5 <br/>\n";
            break 1; /* Sólo sale del switch. */
        case 10:
            echo "En 10; saliendo <br/>\n";
            break 2; /* Sale del switch y del while. */
        default: break;
    }
}
```

continue

continue se utiliza dentro de las estructuras iterativas para saltar el resto de la iteración actual del bucle y continuar la ejecución en la evaluación de la condición, para luego comenzar la siguiente iteración.

continue acepta un argumento numérico opcional, que indica a cuántos niveles de bucles encerrados se ha de saltar al final. El valor por omisión es 1, por lo que salta al final del bucle actual.

Ej:

```
$arr = array('uno', 'dos', 'tres', 'cuatro', 'pare', 'cinco');
foreach ($arr as $val) {
    if ($val == 'pare') {
        continue; /* Imprime todos los valores menos el ‘pare’. */
    }
    echo "$val<br/>\n";
}
```

switch

La sentencia *switch* es similar a una serie de sentencias IF en la misma expresión. En muchas ocasiones, es posible que se quiera comparar la misma variable (o expresión) con muchos valores diferentes, y ejecutar una parte de código distinta dependiendo de a que valor es igual. Para esto es exactamente la expresión *switch*.

Ej: 2 maneras diferentes de realizar lo mismo

```
if ($i == 0) {
    echo "i es igual a 0";
} elseif ($i == 1) {
    echo "i es igual a 1";
} elseif ($i == 2) {
    echo "i es igual a 2";
}

switch ($i) {
    case 0:
        echo "i es igual a 0";
        break;
    case 1:
        echo "i es igual a 1";
        break;
    case 2:
        echo "i es igual a 2";
        break;
}
```

Un caso especial es el *default*. Este caso coincide con cualquier cosa que no se haya correspondido por los otros casos.

Ej:

```
switch ($i) {
    case 0:
        echo "i es igual a 0";
        break;
    case 1:
        echo "i es igual a 1";
        break;
    case 2:
        echo "i es igual a 2";
        break;
    default:
```

```
        echo "i no es igual a 0, 1 ni 2";
}
```

return

return devuelve el control del programa al módulo que lo invoca. La ejecución vuelve a la siguiente expresión después del módulo que lo invoca.

Si se llama desde una función, la sentencia *return* inmediatamente termina la ejecución de la función actual, y devuelve su argumento como el valor de la llamada a la función.

Si se llama desde el ámbito global, entonces la ejecución del script actual se termina. Si el archivo script actual fue incluido o requerido con include o require, entonces el control es pasado de regreso al archivo que hizo el llamado. Además, si el archivo script actual fue incluido con include, entonces el valor dado a *return* será retornado como el valor de la llamada include.

include, require, include_once y require_once

Estas sentencias incluyen y evalúan el archivo especificado.

Cuando se incluye un archivo, el código que contiene hereda el ámbito de las variables de la línea en la cual ocurre la inclusión. Cualquier variable disponible en esa línea del archivo que hace el llamado, estará disponible en el archivo llamado, desde ese punto en adelante. Sin embargo, todas las funciones y clases definidas en el archivo incluido tienen el ámbito global.

Ej:

```
vars.php
<?php
    $color = 'verde';
    $fruta = 'manzana';
?>

test.php
<?php
    echo "Una $fruta $color"; // Una
```

```
    include 'vars.php';
    echo "Una $fruta $color"; // Una manzana verde
?>
```

require es idéntico a include excepto que en caso de fallo producirá un error fatal, en otras palabras, éste detiene el script mientras que include sólo emitirá una advertencia lo cual permite continuar el script.

La sentencia *include_once* incluye y evalúa el fichero especificado durante la ejecución del script. Tiene un comportamiento similar al de la sentencia include, siendo la única diferencia de que si el código del fichero ya ha sido incluido, no se volverá a incluir, e include_once devolverá `TRUE`. Como su nombre indica, el fichero será incluido solamente una vez.

La sentencia *require_once* es idéntica a require excepto que PHP verificará si el archivo ya ha sido incluido y si es así, no se incluye (require) de nuevo.