Tema 4 - Lógica de control en PHP

Índice de contenidos

Bifurcaciones

if … elseif … else

En cuanto una de las condiciones se evalúa como TRUE se ejecuta su bloque de instrucciones asociado y ya no se evalúan las demás.

En una instrucción if se ejecuta como máximo uno de los bloques de instrucciones.

Existen dos sintaxis posibles: de llaves y de dos puntos. Ambas sintaxis producen los mismos resultados.

- Llaves

```
<?php
if ($a > $b) {
echo "a es mayor que b";
} elseif ($a == $b) {
echo "a es igual que b";
} else {
echo "a es menor que b";
}
?>
```

- Dos puntos:

```
<?php
if ($a > $b):
echo $a." es mayor que ".$b;
elseif ($a == $b): // Tenga en cuenta la combinación de las palabras.
echo $a." igual ".$b;
```

```
else:
echo $a." no es ni mayor ni igual a ".$b;
endif;

?>
```

Importante: Hay que tener en cuenta que elseif y else if serán considerados exactamente iguales sólamente cuando se utilizan llaves como en el ejemplo anterior. Al utilizar los dos puntos para definir las condiciones if/elseif, no debe separarse else if en dos palabras o PHP fallará con un error del intérprete.

switch

Sólo se evalúa una expresión. La expresión evaluada no tiene que ser forzosamente de tipo lógico; puede ser numérica o de cadena de caracteres.

Localizado el caso que coincide con el valor de la expresión, se ejecutan las instrucciones de ese caso hasta que se encuentre un break. No hay nada negativo en la no utilización de break dentro de un caso; simplemente hay situaciones en las que conviene utilizarlo y otras en las que no. Por ejemplo, si queremos ejecutar un mismo bloque de instrucciones para varios casos distintos puede venir muy bien no incluir el break. Ejemplo:

```
<?php
switch ($i) {
case 0:
case 1:
case 2: echo "i es menor que 3, pero no negativo"; break;
case 3: echo "i es 3"; break;
default: echo "i es negativo o mayor de 3";
}
?>
```

Si el valor no coincide con ninguno de los casos, se ejecutan las instrucciones asignadas al caso opcional *default*.

Operador ternario ?:

condicion ? valor_si_condicion_TRUE : valor_si_condicion_FALSE

Es un operador, y como tal devuelve un resultado (no ejecuta una instrucción).

Ejemplo correcto:

```
<?php
$edad = 17;
$mensaje = $edad>=18? 'Mayor de edad' : 'Menor de edad';
echo $mensaje;
// También sería válido el siguiente código
echo $edad>=18 ? 'Mayor de edad' : 'Menor de edad'; ?>
```

Ejemplo incorrecto:

```
<?php
$edad = 17;
$edad>=18? echo 'Mayor de edad' : echo 'Menor de edad'; ?>
// Este operador no ejecuta instrucciones
```

Bucles / iteraciones

while

Mientras la evaluación de la condición sea TRUE ejecutará las instrucciones establecidas. Se ejecutará de 0 a N veces.

Existen dos nomenclaturas posibles:

- Con llaves:

```
while (condicion) {
        instrucciones_a_ejecutar_en_cada_iteración
}
```

- Con dos puntos:

```
while (condicion) :
        instrucciones_a_ejecutar_en_cada_iteración
endwhile;
```

do while

Mientras la evaluación de la condición sea TRUE ejecutará las instrucciones establecidas. Se ejecutará de 1 a N veces.

```
do {
        instrucciones_a_ejecutar_en_cada_iteración
} while (condicion)
```

for

```
for (expr_inicial ; expr_continuidad ; expr_post_iteración) {
        instrucciones_de_la_iteración
}
```

Su ejecución se realiza en los siguientes pasos:

1. Se evalúa la expr_inicial.
2. Si expr_continuidad se evalúa como TRUE se ejecuta las instrucciones de la iteración, si no se abandona el bucle.
3. Al terminar cada iteración se ejecuta expr_post_iteración y, se vuelve al paso 2.

Existen dos sintaxis: de llaves y con dos puntos.

- Llaves:

```
for (expr_inicial ; expr_continuidad ; expr_post_iteración) {
        instrucciones_de_la_iteración
}
```

- Dos puntos:

```
for (expr_inicial ; expr_continuidad ; expr_post_iteración) :
        instrucciones_de_la_iteración
endfor;
```

Ejemplo:

```
<?php
for ($x = 0; $x <= 10; $x++) {
  echo "El número es: $x <br>";
}
?>
```

¿Qué hará el siguiente código?

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

foreach

Es un tipo especifico de bucle que se usa para recorrer los elementos de un array.

- Para recorrer claves y valores:

```
foreach ( $miarray as $variable_para_clave => $variable_para_valor) {
        instrucciones_de_la_iteración
}
```

- Para recorrer solamente valores:

```
foreach ( $miarray as $variable_para_valor) {
        instrucciones_de_la_iteración
}
```

Ejemplos:

```php
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
  echo "$value <br>";
}
/* Imprime:
red
green
blue
yellow
*/ ?>
```

```php
<?php
$age = array("Sarah"=>"35", "Ben"=>"37", "Omar"=>"43");

foreach($age as $x => $val) {
  echo "$x = $val<br>";
}
/* Imprime:
Sarah = 35
Peter = 37
Omar = 43
*/
?>

<?php
$age = array("Sarah"=>"35", "Ben"=>"37", "Omar"=>"43");

foreach($age as $val) {
  echo "$val<br>";
}
/* Imprime:
35
37
43
*/
?>
```

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.

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

Importante: En PHP, la sentencia switch se considera una estructura iterativa para continue. Continue se comporta en un switch igual que break (cuando no se proporcionan argumentos). Si un switch está dentro de un bucle, continue 2 continuará con la siguiente iteración del bucle externo.

```php
<?php
foreach ($arr as $clave => $valor) {
    // saltar los miembros pares
    if (!($clave % 2)) {
        continue;
    }
    hacer_algo($valor);
}

$i = 0;
while ($i++ < 5) {
    echo "Exterior<br />\n";
    while (1) {
        echo "Medio<br />\n";
        while (1) {
            echo "Interior<br />\n";
            continue 3;
        }
        echo "Esto nunca se imprimirá.<br />\n";
    }
    echo "Ni esto tampoco.<br />\n";
}
```

```
?>
```

Ejemplo de break con continue:

```
<?php
$stack = array('first', 'second', 'third', 'fourth',
'fifth');

foreach($stack AS $v){
     if($v == 'second')continue;
     if($v == 'fourth')break;
     echo $v.'<br>';
}

/* Imprime:
first
third
*/
```

Funciones definidas por el usuario

Las funciones pueden recibir argumentos o parámetros y pueden devolver como máximo un valor con **return**, que puede ser un valor simple, array u objeto entre otros.

Los nombres de las funciones no deben empezar por $, su primer carácter debe ser una letra o un signo de subrayado, y a continuación pueden usar cualquier combinación de letras, números o signos de subrayado.

PHP es case-sensitive con los nombres de las funciones definidas por el usuario.

```php
<?php
function familyName($fname) {
  echo "$fname García.<br>";
}

familyName("Luisa");
familyName("Christian");
familyName("Yarei");
?>
```

En una función puede haber **varios return**. Con return, además de devolver un dato, se termina la ejecución de la función.

```php
<?php
function determinarResultado($numero) {
    if ($numero > 0) {
        return "El número es positivo";
    } elseif ($numero < 0) {
        return "El número es negativo";
    } else {
        return "El número es cero";
    }
}

$resultado = determinarResultado(5);
echo $resultado; // Muestra "El número es positivo"
$resultado = determinarResultado(-3);
echo $resultado; // Muestra "El número es negativo"
$resultado = determinarResultado(0);
echo $resultado; // Muestra "El número es cero"
?>
```

Las funciones presentan las siguientes ventajas:

- Ahorro de escritura: Nos evita reescribir una y otra vez el mismo código.
- Facilidad de mantenimiento: Nos facilita las labores de mantenimiento al tener concentradas en una única función instrucciones que se utilizan en distintos puntos del script.
- Podemos crear funciones que reciban un número variable de argumentos.
- Podemos asignar valores predeterminados para los argumentos que no se incluyan al solicitar la ejecución de la función.

Por defecto, los datos que recibe una función como argumentos y el que devuelve mediante return se asignan por valor, pero también pueden asignarse por referencia.

```php
<?php
// Función que toma un valor por referencia y lo modifica
function modificarPorReferencia(&$valor) {
    $valor *= 2;
}
// Variable
$numero = 5;
echo "Antes de llamar a la función: $numero<br>";
//Llama a la función pasando la variable por referencia
modificarPorReferencia($numero);
//Ahora el valor de $numero será 10
echo "Después de llamar a la función: $numero<br>";
?>
```

Recursividad en funciones

PHP admite recursividad en las funciones , es decir, que desde dentro de la función se efectúe una llamada a la propia función. Hay que tener cuidado con las

funciones recursivas, pues un abuso en el número de niveles de anidamiento de la recursividad puede provocar un colapso del sistema.

```php
<?php
function factorial($n) {
    if ($n <= 1) {
        return 1;
    } else {
        return $n * factorial($n - 1);
    }
}

// Calcular el factorial de 5
$resultado = factorial(5);
echo "El factorial de 5 es $resultado";
?>
```

Función en cadena de caracteres

El nombre de una función puede ser una cadena de caracteres almacenada en una variable.

```php
<?php
// Definir una cadena que contiene el nombre de la función
$nombreFuncion = "miFuncion";

// Definir la función que queremos llamar
function miFuncion($mensaje) {
    echo "La función miFuncion se ha llamado con el mensaje: $mensaje";
}
```

```php
/* Llamar a la función utilizando la variable que contiene el nombre */
$mensaje = "Hola desde la función";
$nombreFuncion($mensaje);
/* Imprime: La función miFuncion se ha llamado con el mensaje: Hola desde la función */
?>
```

Funciones con un número variable de argumentos

Argumentos por defecto y opcionales

Los argumentos opcionales siempre deben colocarse a la derecha de los obligatorios. El valor predeterminado debe ser explícitamente un dato; no puede ser una expresión ni una variable.

Un argumento opcional no puede dejarse desierto en la llamada de la función para intentar especificar otros ubicados a su derecha. No es válido llamar a la función con funcion(arg1 , , arg3);

```php
<?php
/* Función con un parámetro obligatorio y otro con valor por defecto */
function saludar($nombre, $saludo = "Hola") {
    echo "$saludo, $nombre!";
}
// Llamar a la función con ambos parámetros
saludar("Juan"); // Esto imprimirá "Hola, Juan!"

/* Llamar a la función proporcionando un valor personalizado para $saludo */
saludar("Maria", "Buenos días");
```

```
// Esto imprimirá "Buenos días, Maria!"
?>
```

Ejemplo incorrecto:

```
<?php
function haceryogurt($contenedor = "bowl", $sabor)
{
    return "Preparando un $contenedor de yogur de $sabor.\n";
}

echo haceryogurt("frambuesa");
// "frambuesa" es $contenedor, no $sabor
// Este ejemplo es INCORRECTO
?>
```

Argumentos no especificados (func_get_args())

Las funciones de PHP pueden recibir un número variable de argumentos para los que no se dispone de un valor predeterminado.

En estos casos es de interés el uso de las siguientes funciones (método cada vez más en desuso):

func_num_args()

Devuelve el número total de argumentos que ha recibido la función.

func_get_arg(posicion)

Devuelve el dato del argumento recibido por la función en la posición indicada; el primero de ellos corresponde a la posición 0.

func_get_args()

Devuelve un array con todos los argumentos recibidos por la función.

```
<?php
```

```php
function hacerMacedonia()
{
    $numFrutas = func_num_args();
    $frutas = func_get_args();

    if ($numFrutas === 0) {
        return "No hay frutas para hacer la macedonia.";
    }
    $macedonia = "Macedonia de frutas con: ";
/* Nota: es mejor recorrer el array con un foreach, pero esto
es un ejemplo del método func_get_arg (posición)
Con foreach:
    foreach ($frutas as $fruta) {
        $macedonia .= $fruta;
        if ($fruta !== end($frutas)) {
            $macedonia .= ", ";
        }
    }
*/
    for ($i = 0; $i < $numFrutas; $i++) {
        $macedonia .= func_get_arg($i);
        if ($i < $numFrutas - 1) {
            $macedonia .= ", ";
        }
    }

    return $macedonia;
}

echo hacerMacedonia("manzana", "plátano", "uva", "kiwi");
echo hacerMacedonia();
?>
```

Argumentos no especificados (sintaxis …)

Las listas de argumentos pueden incluir el token **...** para indicar que la función acepta un número variable de argumentos. Los argumentos se pasarán a la variable proporcionada como un array.

```
<?php
function sumaVariable(...$numeros) {
    $resultado = 0;
    foreach ($numeros as $numero) {
        $resultado += $numero;
    }
    return $resultado;
}
echo sumaVariable(2, 3, 5, 7); // Salida: 17
?>
```

También pueden usarse los … para descomprimir un array en el paso de parámetros

```
<?php
function add($a, $b) {
return $a + $b;
}

echo add(...[1, 2]); //Imprime 3

$a = [1, 5];
echo add(...$a); //Imprime 6
```

```php
$b = [2, 8, 1, 23];
echo add(...$b); //Imprime 10

$c = [7];
echo add(...$c); //No imprime nada
?>
```

Para ampliar el tema (en inglés):
https://www.php.net/manual/en/functions.arguments.php#functions.variable-arg-list

Incluir código con include y require

Sirven para incluir el código de un archivo dentro de otro. Si no se encuentra el archivo:

include genera **aviso**

require genera **error**

include(ruta_de_archivo);

require(ruta_de_archivo);

La ruta puede expresarse:

- Explícitamente: De forma absoluta o relativa (usando . para el directorio actual y .. para el directorio padre).

- Implícitamente: Si escribimos simplemente el nombre de un archivo (sin ruta), PHP intentará localizarlo buscando en el siguiente orden (en cuanto lo encuentre en uno de estos destinos no sigue buscando en los demás):
    1. En la carpeta especificada en la directiva include_path del **php.ini**,
    2. En el propio directorio donde se encuentra el archivo php que ha ordenado la inclusión.

include y require interrumpen el intérprete PHP. Para que el código incluido se interprete como PHP debe tener sus propio elementos <?php y ?>

El código incluido adquiere el ámbito de la posición en la que es incrustado, pero las constantes mágicas se evalúan antes de la inclusión.

Ejemplo básico de include:

Archivo vars.php

```php
<?php
$color = 'verde';
$fruit = 'manzana';
?>
```

Archivo test.php

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

Ejemplo de include dentro de una función:

```php
<?php
function foo()
{
    global $color;
    include 'vars.php';
    echo "Una $fruit $color"; // Una manzana verde
}
```

```php
/* vars.php está en el ámbito de foo() por lo que
$fruit NO está disponible fuera del ámbito de la
función. $color sí lo está porque la declaramos
global */

foo(); // Una manzana verde
echo "Una $fruit $color"; // Una verde
?>
```