Arrays y estructuras de control

1. ARRAYS

Aunque los tipos de datos básicos son muy útiles, en ocasiones necesitarás disponer de tipos más complejos.

En PHP se pueden utilizar **arrays**. Un array es una colección indexada de elementos.

A diferencia de otros lenguajes, en PHP cada uno de esos elementos puede ser de un tipo de datos distinto. Esto es un efecto de que los arrays no tengan que declararse, como ocurre con las variables.

Piensa en un array como en una lista de elementos. Puedes acceder a cada uno de sus miembros a partir de un determinado **índice** que indica su posición en la lista.

Observa el siguiente código PHP:

```
<?php
      $a0 = 0;
      $a1 = 1;
      $a2 = 2;
      $a3 = 3;
      $a4 = 4;
      $a5 = 5;
      $suma = $a0 + $a1 + $a2 + $a3 + $a4 + $a5;
      echo "<p>$suma</p>";
?>
```

Como puedes comprobar, se utilizan 6 variables enteras, se indican sus valores y finalmente se escribe en la página la suma de todas ellas.

Aunque el código es correcto, se podría utilizar un array de 6 elementos en lugar de utilizar las 6 variables independientes. Esto, además de reducir el código, es una forma de indicar que los elementos están relacionados.

Hay distintas formas de crear un array. La forma más sencilla es hacerlo como si se tratara de una variable más pero incluyendo corchetes tras el nombre.

Por ejemplo, con la línea **$a[0] = 0;** se creará un array con un único elemento, cuya posición es la 0 ya que los elementos de los arrays empiezan por este índice.

La única diferencia que vemos es que tras el nombre de la variable se indica, entre corchetes, el índice del elemento del array.

```
$a[1] = 1;
$a[2] = 2;
...
```

A medida que vamos estableciendo el valor de otros índices del array, estamos creando elementos del mismo.

Este código se completaría con las líneas necesarias para tener un array de 6 elementos, cuyos índices serían del 0 al 5.

Sin embargo, hay una forma más cómoda de hacerlo:

```
$a = array(0, 1, 2, 3, 4, 5);
```

Con una única línea hemos creado un array de 6 elementos y hemos dado el valor de cada uno de ellos.

Como te había comentado, en PHP cada elemento del array puede ser de un tipo distinto. Así que nadie nos impide guardar, por ejemplo, una cadena de texto en el segundo elemento (`$a[1] = "hola";`).

Fíjate ahora cómo quedaría la suma:

```
$suma2 = $a[0] + $a[1] + $a[2] + $a[3] + $a[4] + $a[5];
```

Ahora obtenemos en la variable **$suma2** la suma de todos los elementos del array. Para ello hemos accedido a cada uno de esos elementos indicando su índice entre los corchetes. Por lo tanto **$a[0]** es el primer elemento del array, **$a[1]** es el segundo, etc.

A partir de PHP 5.4 también se puede usar la sintaxis de array corta para crear un array, que reemplaza el operador array() por [ ].

En el ejemplo que estamos viendo, quedaría así:

$a = [0, 1, 2, 3, 4, 5];

2. ESTRUCTURAS DE REPETICIÓN

PHP incorpora **estructuras de control** que permiten controlar el flujo de ejecución de un programa. Si no introducimos este tipo de estructuras, el código se ejecuta secuencialmente, tal como está escrito, es decir, de izquierda a derecha y de arriba abajo.

Sin embargo, en ocasiones desearás ejecutar una u otra instrucción según una condición dada o repetir un conjunto de instrucciones en más de una ocasión. Para ello, se utilizan las **estructuras de control**.

Existen dos categorías de estructuras de control: estructuras de **decisión** y estructuras de **repetición**.

Empezaremos estudiando este último tipo de estructuras, también conocidas como **bucles**.

Por ejemplo, imagina que quieres repetir un **número determinado de veces** un conjunto de instrucciones o hacerlo **mientras** o **hasta que se cumpla cierta condición**.

En el primer caso podrás utilizar la estructura de repetición **for**, que presenta la siguiente sintaxis:

```
for (expr1; expr2; expr3)
{
    instrucciones
}
```

Donde **expr1** se evalúa al principio del bucle. Normalmente es la asignación de valor inicial a una variable.

En cada iteración se evalúa **expr2**, que es la expresión que determina si se tiene que volver a entrar en el bucle o no. Por su parte, **expr3** determina cómo se modifica la variable contador. Además, en cada iteración se ejecuta el conjunto de instrucciones de su cuerpo.

Con un ejemplo se verá más fácil.

La expresión $i++ es equivalente a $i = $i + 1.

```
$suma2 = 0;
for ($i = 0; $i <= 5; $i++)
{
     $suma2 += $a[$i];
}
```

Veamos algunos detalles interesantes de este código. La idea principal es la misma que teníamos antes, donde la variable **$suma2** guardaba la suma de todos los elementos del array **$a**.

Primero damos un valor inicial a **$suma2**, que en este caso es **0**. Después utilizamos un bucle **for** para que se repita desde que **i** vale **0** hasta que valga **5** (como indica la expresión **$i <= 5**).

En cada iteración **i** aumenta de uno en uno, como se establece con la expresión **$i++**.

Podría utilizarse cualquier expresión para indicar cómo se modifica el valor de la variable **$i**. Por ejemplo, si queremos que disminuya en cada iteración, la última expresión del interior de los paréntesis podría ser **$i = $i - 2** para que disminuya de dos en dos.

Finalmente, la única instrucción del cuerpo del bucle es la correspondiente a que se sume el elemento **i** oportuno. En cada iteración **$i** tomará un valor, que será el que se sumará.

Se utiliza el operador += como atajo para escribir lo siguiente:

$suma2 = $suma2 + $a[$i];

Otros operadores de asignación son:

Operador	**Ejemplo**	**Equivale a**
+=	$a += 2	$a = $a + 2 (suma)
-=	$a -= 2	$a = $a – 2 (diferencia)
*=	$a *= 2	$a = $a * 2 (producto)
/=	$a /= 2	$a = $a / 2 (división)
%=	$a %= 2	$a = $a % 2 (resto de división)
.=	$a .= "hola"	$a = $a . "hola" (concatenación)

Observa que cada operador no es más que la combinación del operador básico de asignación (=) y un operador aritmético (suma, diferencia, producto, etc.) o de concatenación.

Como curiosidad, el operador de división / devuelve un valor flotante en todos los casos, incluso si los dos operandos son enteros.

Además, vemos que para encerrar un bloque de instrucciones, se utilizan las llaves. Si solo es una instrucción (como en este caso), no es necesario, pero sí conveniente para que el código sea más claro.

Aunque el bucle **for** está pensado para utilizarse cuando se conoce de antemano el número de veces que debe repetirse, es posible utilizar la instrucción **break** para salir del bucle anticipadamente.

Por ejemplo, podríamos comprobar si se cumple cierta condición y decidir salir del bucle aunque el valor inicial no haya llegado al valor final.

En otras ocasiones, sin embargo, desearás repetir un conjunto de instrucciones **mientras** se cumpla una determinada condición o **hasta** que se cumpla otra. Para ello, se puede utilizar el bucle **while**.

La sintaxis del bucle **while** es la siguiente:

```
while (expr)
{
   instrucciones
}
```

El significado de este bucle es muy sencillo. Se indica que tienen que ejecutarse las instrucciones del cuerpo del bucle mientras **expr** se cumpla (es decir, se evalúe a TRUE). Fíjate que, en este caso, la expresión se evalúa siempre al principio, por lo que si la primera vez no se cumple, el cuerpo del bucle no llegará a ejecutarse.

Por ejemplo, vamos a calcular la suma de los elementos del array utilizando un bucle **while**:

```
$suma2 = 0;
$i = 0;
while ($i <= 5)
{
     $suma2 += $a[$i];
}
```

Bien, espero que te hayas dado cuenta de que falta algo muy importante. Fíjate que en el cuerpo del bucle no hay ninguna línea que modifique el valor de la variable **$i**, que es la que determina si se repite el bucle o no.

En este caso, la variable siempre valdría su valor inicial (0) por lo que el bucle se repetiría para siempre (es lo que se conoce como un bucle infinito), ya que siempre se cumpliría que **$i** es menor o igual que **5**.

Así que la forma correcta sería:

```
$suma2 = 0;
$i = 0;
while ($i <= 5)
{
     $suma2 += $a[$i];
     $i++;
}
```

En estos bucles también se puede utilizar la instrucción break para salir anticipadamente.

Existe una versión distinta del bucle **while** en la que la expresión se evalúa al final de cada iteración del bucle.

Su sintaxis es:

```
do
{
   instrucciones
} while (expr)
```

Fíjate que, como en este caso la expresión se evalúa al final, es seguro que al menos una vez se ejecutará el conjunto de instrucciones del cuerpo del bucle.

3. ESTRUCTURAS DE DECISIÓN

Otro tipo de estructuras de control que se utiliza frecuentemente son las **estructuras de decisión**. En este caso, el propósito es ejecutar un conjunto de instrucciones u otro según una determinada condición.

La estructura clásica de decisión es:

```
if (condición)
{
   instrucciones
}
```

Debes entender esta estructura de la siguiente forma: **si** se cumple la condición, **entonces** ejecutar las instrucciones.

```
<?php
     $dia = 1;
     if ($dia == 1)
           $resultado = "lunes";
     if ($dia == 2)
           $resultado = "martes";
     if ($dia == 3)
           $resultado = "miércoles";
     if ($dia == 4)
           $resultado = "jueves";
     if ($dia == 5)
           $resultado = "viernes";
     if ($dia == 6)
           $resultado = "sábado";
     if ($dia == 7)
           $resultado = "domingo";
     echo "<p>Hoy es <b>$resultado</b>.</p>";
?>
```

En el código anterior damos un valor a una variable entera y después comprobamos dicho valor con varias estructuras **if**.

Así, si el valor de **$dia** es **1**, entonces se ejecuta la instrucción **$resultado = "lunes"**; si el valor es **2**, se ejecuta **$resultado = "martes"**, etc.

Al final, se imprime en la página web la frase *"Hoy es ...."*, concatenando el día que hemos obtenido como resultado tras las estructuras de decisión.

Tal como está escrito este código, se realizan 7 comprobaciones del valor de la variable **$dia**.

Una extensión de esta estructura es:

```
if (condición1)
{
  instrucciones1
}
elseif (condición2)
{
   instrucciones2
}
else
{
   instruccionesN
}
```

En este caso, si se cumple la **condición1**, se ejecuta el conjunto de **instrucciones1** y, en caso contrario, se comprueba el resto de condiciones hasta que se cumpla alguna, ejecutándose el correspondiente conjunto de instrucciones, o hasta que se llegue a la cláusula **else**.

Esto puede servir para anidar varias estructuras **if**, es decir, introducir una estructura en el interior de otra. Obsérvalo:

```
<?php
     $dia = 1;
     if ($dia == 1)
     {
           $resultado = "lunes";
     }
     elseif ($dia == 2)
     {
           $resultado = "martes";
     }
     elseif ($dia == 3)
     {
           $resultado = "miércoles";
     }
```

En este ejemplo no serían necesarias las llaves { y } ya que solo engloban una instrucción. Sin embargo, es una buena costumbre hacerlo.

```
      elseif ($dia == 4)
      {
            $resultado = "jueves";
      }
      elseif ($dia == 5)
      {
            $resultado = "viernes";
      }
      elseif ($dia == 6)
      {
            $resultado = "sábado";
      }
      else
      {
            $resultado = "domingo";
      }
      echo "<p>Hoy es <b>$resultado</b>.</p>";
?>
```

Observa la utilización de la cláusula **elseif** para introducir una nueva condición. De esta forma, solo se comprobará el valor de la variable **$dia** las veces necesarias.

Además, la instrucción correspondiente a la cláusula **else** solo se ejecutará cuando no se cumpla ninguna de las condiciones anteriores.

De hecho, es habitual utilizar la estructura **if - else**, en la que tenemos únicamente una cláusula **if** y únicamente una cláusula **else**.

Aunque anidar varias cláusulas **elseif** permite reducir el número de comprobaciones que se deben realizar, existe una estructura de decisión más apropiada para estos casos, sobre todo para añadir claridad al código.

Esta estructura es:

```
switch (expresiónDeComparación)
{
   case listaExpresiones1:
      instrucciones1
   case listaExpresiones2:
      instrucciones2
......
   case listaExpresionesN:
      instruccionesN
   default:
      instruccionesN+1
}
```

Esta instrucción evalúa solo una vez la expresión de comparación y ejecuta el bloque de instrucciones de la primera cláusula **case** cuya **listaExpresiones** cumpla con ella.

Es importante no olvidar introducir la instrucción **break** al final del bloque de las instrucciones de cada **case**. Esto es así porque PHP ejecutará el grupo de instrucciones que aparece a partir de la primera cláusula **case** que se cumpla hasta encontrar un **break**.

```
switch ($dia)
{
     case 1:
           $resultado = "lunes";
           break;
     case 2:
           $resultado = "martes";
           break;
     case 3:
           $resultado = "miércoles";
           break;
     case 4:
           $resultado = "jueves";
           break;
     case 5:
           $resultado = "viernes";
           break;
     case 6:
           $resultado = "sábado";
           break;
     case 7:
           $resultado = "domingo";
           break;
     default:
           $resultado = "día incorrecto";
}
```

Por lo tanto, si **$dia** vale **1** y no escribimos las instrucciones **break**, entonces se ejecutaría la asignación de la cláusula **case 1** pero también de las demás.

Con la cláusula **default** indicamos qué debe ocurrir cuando ninguna de las condiciones anteriores se ha cumplido. No es obligatorio hacerlo.

Las listas de expresiones de las cláusulas **case** pueden ser más complejas. Así, podría referirse a más de un valor, a un rango de valores, etc.

Además, esta estructura tiene como ventaja respecto a **if** que las condiciones que se evalúan no tienen por qué ser únicamente TRUE o FALSE, sino que también podrían dar como resultado un valor entero, decimal o incluso una cadena de texto.

4. COMBINAR ESTRUCTURAS

Vamos a practicar con todo lo que hemos visto en la lección: utilizaremos arrays y combinaremos estructuras de repetición y de decisión.

El objetivo de nuestro código será la búsqueda de un determinado valor en un array de enteros. Si el valor a buscar se encuentra en el array, entonces imprimiremos en pantalla un mensaje indicando esa circunstancia.

```
<?php
      $a = array(2, 4, 1, 3, 15, 7, 8, 9, 22, 6);
      $valor = 15;
      $encontrado = FALSE;
      for ($lugar = 0; $lugar <= 9; $lugar++)
      {
            if ($a[$lugar] == $valor)
            {
                  $encontrado = TRUE;
                  break;
            }
      }
      if ($encontrado)
      {
            echo "El valor $valor está en la lista.";
      }
      else
      {
            echo "El valor $valor no está en la lista.";
      }
?>
```

- Lo primero es crear el array de enteros para lo que utilizamos la línea:

```
$a = array(2, 4, 1, 3, 15, 7, 8, 9, 22, 6);
```

De esta forma creamos un array de **10** enteros.

- Utilizamos la variable entera **$valor** para guardar el valor a buscar. Inicialmente le damos el valor **15**, que, como puedes comprobar, sí que está incluido en el array.
- La variable **$encontrado** será donde guardemos si el valor se encuentra o no en el array. Inicialmente se establece a **FALSE**, con lo que indicamos que el valor a buscar no se encuentra en el array.
- El bucle for nos permite recorrer el array e ir comprobando si el valor buscado se encuentra en él. Fíjate en la forma que utilizamos para acceder a un elemento del array **$a[$lugar]** y la presencia de la estructura **if** en el interior del bucle.

- En el caso de que se encuentre el valor buscado, se establece **$encontado = TRUE** y se sale del bucle anticipadamente con **break**, ya que no es necesario seguir buscando más.
- Finalmente, debemos imprimir en pantalla el resultado de la búsqueda, por lo que comprobamos el valor de la variable **$encontrado** tras recorrer el array con el bucle **for**.

  Si se da la circunstancia de que su valor es TRUE, entonces se imprime un mensaje en la página web indicando que el valor está en la lista.

De la misma forma, la expresión (!$encontrado) es equivalente a ($encontrado == FALSE).

- Observa que la expresión **($encontrado)** es equivalente a **($encontrado == TRUE)**.