Unidad 5. Arrays y Funciones en PHP

Arrays clásicas

Un array es un tipo de dato capaz de almacenar múltiples valores. Se utiliza cuando tenemos muchos datos parecidos, por ejemplo, para almacenar la temperatura media diaria en Málaga durante el último año podríamos utilizar las variables `$temp0`, `$temp1`, `$temp2`, `$temp3`, `$temp4`, … y así hasta 365 variables distintas pero sería poco práctico; es mejor utilizar un array de nombre `$temp` y usar un índice para referirnos a una temperatura concreta.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $temp[0] = 16;
      $temp[1] = 15;
      $temp[2] = 17;
      $temp[3] = 15;
      $temp[4] = 16;
      echo "La temperatura en Málaga el cuarto día del año fue de ";
      echo $temp[3], "ºC";
    ?>
  </body>
</html>
```

Los valores de un array se pueden asignar directamente en una línea. El índice comienza en 0.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $temp = array(16, 15, 17, 15, 16);
      echo "La temperatura en Málaga el cuarto día del año fue de ";
      echo $temp[3], "ºC";
    ?>
  </body>
</html>
```

En este otro ejemplo, asignamos valores aleatorios a los elementos de un array.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      echo "<b>Notas:</b><br>";
      for ($i = 0; $i < 10; $i++) {
        // números aleatorios entre 0 y 10 (ambos incluidos)
        $n[$i] = rand(0, 10);
      }

      foreach ($n as $elemento) {
        echo $elemento, "<br>";
      }
    ?>
  </body>
</html>
```

A partir de PHP 5.4 se puede utilizar la sintaxis abreviada (utilizando corchetes) para definir un array.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $color = ["verde", "amarillo", "rojo", "azul", "blanco", "gris"];
      echo "Mañana me pongo una camiseta de color ", $color[rand(0, 5)], ".";
    ?>
  </body>
</html>
```

Como puedes ver en los ejemplos anteriores, no es necesario definir previamente un array antes de utilizarlo. Tampoco hay límite en cuanto al número de elementos que se pueden añadir al mismo. No obstante, se pueden crear arrays de tamaño fijo con `SplFixedArray` que son más eficientes en cuanto a uso de la memoria y más rápidas en las operaciones de lectura y escritura.

La función `var_dump()` se utiliza para mostrar el tipo y el valor de un dato, en este caso muestra los tipos y valores de cada uno de los elementos del array.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = new SplFixedArray(10);
      $a[0] = 843;
      $a[2] = 11;
      $a[6] = 1372;
      // Los valores del array que no se han inicializado son NULL
      foreach ($a as $elemento) {
        var_dump($elemento);
        echo "<br>";
      }
    ?>
  </body>
</html>
```

Arrays asociativos

En un array asociativo se pueden utilizar índices que no son numéricos, a modo de claves. Veamos un ejemplo de un array asociativo que almacena alturas (en centímetros) y que como índice o clave utiliza nombres.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura = array("Rosa" => 168, "Ignacio" => 175, "Daniel" => 172, "Rubén" => 182);
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm";
    ?>
  </body>
</html>
```

También se pueden asignar los valores de forma individual:

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura['Rosa'] = 168;
      $estatura['Ignacio'] = 175;
      $estatura['Daniel'] = 172;
      $estatura['Rubén'] = 182;
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm";
    ?>
  </body>
</html>
```

Si el lector es suficientemente perspicaz ya se habrá dado cuenta de que `$_GET` y `$_POST` son arrays asociativos. Con los arrays asociativos se puede usar también la sintaxis abreviada que emplea corchetes.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura = [
        "Rosa" => 168,
        "Ignacio" => 175,
        "Daniel" => 172,
        "Rubén" => 182,
      ];
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm";
    ?>
  </body>
</html>
```

Arrays bidimensionales

Un array bidimensional utiliza dos índices para localizar cada elemento. Podemos ver este tipo de datos como un array que, a su vez, contiene otros arrays. En el siguiente ejemplo se define un array con 4 elementos que, a su vez, son un array asociativo cada uno de ellos.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <?php
      $persona = array (
        array( "nombre" => "Rosa", "estatura" => 168, "sexo" => "F"),
        array( "nombre" => "Ignacio", "estatura" => 175, "sexo" => "M"),
        array( "nombre" => "Daniel", "estatura" => 172, "sexo" => "M"),
        array( "nombre" => "Rubén", "estatura" => 182, "sexo" => "M")
      );

      echo "<b>DATOS SOBRE EL PERSONAL<b><br><hr>";

      $numPersonas = count($persona);

      for ($i = 0; $i < $numPersonas; $i++) {
        echo "Nombre: <b>", $persona[$i]['nombre'], "</b><br>";
        echo "Estatura: <b>", $persona[$i]['estatura'], " cm</b><br>";
        echo "Sexo: <b>", $persona[$i]['sexo'], "</b><br><hr>";
      }
    ?>
  </body>
</html>
```

Observa que la función `count()` permite saber el número de elementos de un array.

Iterador `foreach`

El iterador `foreach` se utiliza para recorrer todos los elementos de un array sin que tengamos que preocuparnos por los índices ni por el tamaño del array.

Es común cometer errores al utilizar arrays por no establecer bien el valor inicial o el valor final en el bucle que la recorre, o por no determinar bien el tamaño. Con `foreach` nos podemos despreocupar de todo eso, simplemente recorremos todo el array de principio a fin. Veamos un ejemplo.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $cajonDeSastre = [30, -7, "Me gusta el queso", 56, "¡eh!", 237];

      foreach ($cajonDeSastre as $cosa) {
        echo "$cosa<br>";
      }
    ?>
  </body>
</html>
```

Cómo recoger datos para un array mediante un formulario

Imagina que quieres pedir seis números por teclado y quieres guardar esos números en un array. Se puede pedir un número mediante un formulario, a continuación el siguiente, luego el siguiente, etc. pero ¿cómo enviarlos? Hay un truco muy sencillo. Se pueden ir concatenando valores en una cadena de caracteres y el resultado de esa concatenación se puede reenviar una y otra vez en un formulario como campo oculto. Por último, se puede utilizar la función `explode()` para convertir la cadena de caracteres en un array.

Es importante señalar que los valores que se van concatenando deben tener algún tipo de separación dentro de la cadena, por ejemplo un espacio en blanco.

A continuación se muestra un ejemplo completo:

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $n = $_GET['n'];
      $contadorNumeros = $_GET['contadorNumeros'];
      $numeroTexto = $_GET['numeroTexto'];

      if (!isset($n)) {
        $contadorNumeros = 0;
        $numeroTexto = "";
      }

      // Muestra los números introducidos
```

```
      if ($contadorNumeros == 6) {
        $numeroTexto = $numeroTexto . " " . $n; // añade el último número leído
        $numeroTexto = substr($numeroTexto, 2); // quita los dos primeros
                                                // espacios de la cadena
        $numero = explode(" ", $numeroTexto);
        echo "Los números introducidos son: ";
        foreach ($numero as $n) {
          echo $n, "  ";
        }
      }

      // Pide número y añade el actual a la cadena
      if (($contadorNumeros < 6) || (!isset($n))) {
        ?>
        <form action="#" method="get">
          Introduzca un número:
          <input type="number" name ="n" autofocus>
          <input type="hidden" name="contadorNumeros" value="<?= ++$contadorNumeros ?>">
          <input type="hidden" name="numeroTexto" value="<?= $numeroTexto . " " . $n ?>">
          <input type="submit" value="OK">
        </form>
        <?php
      }
    ?>
  </body>
</html>
```

Implementando funciones para reutilizar código

En programación es muy frecuente reutilizar código, es decir, usar código ya existente. Cuando una parte de un programa requiere una funcionalidad que ya está implementada en otro programa no tiene mucho sentido emplear tiempo y energía en implementarla otra vez.

Una función es un trozo de código que realiza una tarea muy concreta y que se puede incluir en cualquier programa cuando hace falta resolver esa tarea. Opcionalmente, las funciones aceptan una entrada (parámetros de entrada) y devuelven una salida.

Observa el siguiente ejemplo. Se trata de un programa que pide un número mediante un formulario y luego dice si el número introducido es o no es primo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
```

```
      $n = $_POST['n'];

      if (!isset($n)) {
      ?>
        Introduzca un número para saber si es primo o no.<br>
        <form action=numeroPrimo.php method="post">
          <input type="number" name="n" min="0" autofocus><br>
          <input type="submit" value="Aceptar">
        </form>
      <?php
      } else {
        $esPrimo = true;

        for ($i = 2; $i < $n; $i++) {
          if ($n % $i == 0) {
            $esPrimo = false;
          }
        }

        // El 0 y el 1 no se consideran primos
        if (($n == 0) || ($n == 1)) {
          $esPrimo = false;
        }

        if ($esPrimo) {
            echo "El $n es primo.";
        } else {
            echo "El $n no es primo.";
        }
      }
    ?>
  </body>
</html>
```

Podemos intuir que la tarea de averiguar si un número es o no primo será algo que utilizaremos con frecuencia más adelante así que podemos aislar el trozo de código que realiza ese cometido para usarlo con comodidad en otros programas.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php

      // Programa principal ////////////////////////////////////////////////

      $numero = $_POST['numero'];
```

```
    if (!isset($numero)) {
    ?>
      Introduzca un número para saber si es primo o no.<br>
      <form action=numeroPrimoConFuncion.php method="post">
        <input type="number" name="numero" min="0" autofocus><br>
        <input type="submit" value="Aceptar">
      </form>
    <?php
    } else {
      if (esPrimo($numero)) {
          echo "El $numero es primo.";
      } else {
          echo "El $numero no es primo.";

      }
    }

    // Funciones ////////////////////////////////////////////////////////

    function esPrimo($n) {
      $esPrimo = true;

      for ($i = 2; $i < $n; $i++) {
        if ($n % $i == 0) {
          $esPrimo = false;
        }
      }

      // El 0 y el 1 no se consideran primos
      if (($n == 0) || ($n == 1)) {
        $esPrimo = false;
      }

      return $esPrimo;
    }
  ?>
  </body>
</html>
```

Cada función tiene una cabecera y un cuerpo. En el ejemplo anterior la cabecera es

```
function esPrimo($n)
```

La función `esPrimo()` toma como parámetro un número entero y devuelve un valor lógico (`true` o `false`). Observa que, a diferencia de otros lenguajes de programación fuertemente tipados como Java, en PHP no hace falta indicar el tipo de dato que devuelve la función ni el/los tipo/s de datos de los parámetros que se pasan.

Creación de bibliotecas de funciones

Por lo general y salvo casos puntuales, las funciones se suelen agrupar en ficheros. Estos ficheros de funciones se incluyen posteriormente en el programa principal mediante `include` o `include_once` seguido del nombre completo del fichero.

Veamos cómo utilizar la función esPrimo() desde un fichero independiente. El siguiente código corresponde al fichero matematicas.php.

```php
<?php

  function esPrimo($n) {
    $esPrimo = true;

    for ($i = 2; $i < $n; $i++) {
      if ($n % $i == 0) {
        $esPrimo = false;
      }
    }

    // El 0 y el 1 no se consideran primos
    if (($n == 0) || ($n == 1)) {
      $esPrimo = false;
    }

    return $esPrimo;
  }
```

El programa principal es el siguiente.

```php
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // Carga las funciones matemáticas
      include 'matematicas.php';

      $numero = $_POST['numero'];

      if (!isset($numero)) {
      ?>
        Introduzca un número para saber si es primo o no.<br>
        <form action=numeroPrimo2.php method="post">
          <input type="number" name="numero" min="0" autofocus><br>
          <input type="submit" value="Aceptar">
```

```
        </form>
      <?php
      } else {
        if (esPrimo($numero)) {
            echo "El $numero es primo.";

        } else {
            echo "El $numero no es primo.";
        }
      }
    ?>
  </body>
</html>
```

¿Se pueden sobrecargar las funciones en PHP?

En la mayoría de lenguajes de programación, las funciones se pueden sobrecargar. Esto significa que se pueden definir varias funciones con el mismo nombre pero con distinto número de parámetros, o bien con el mismo número de parámetros pero de distinto tipo.

En PHP no se pueden sobrecargar funciones, es decir, no podemos definir dos funciones con el mismo nombre aunque tengan distinto número de parámetros.