6. Funciones

6.1 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.

6.2 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>
```

6.3 ¿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; pero sí se puede hacer "un apaño" para simular el comportamiento de la sobrecarga.

Fíjate en el siguiente código.

```
<?php
  // Ejemplo de sobrecarga de un método según el
  // número de parámetros que se pasan.
  //
  // Si se pasa un número, se devuelve el cuadrado;
  // si se pasan dos, se devuelve la multiplicación
  // y si se pasan tres, se devuelve la suma.
  function opera($x, $y, $z) {
    if (!isset($y)) {
      return $x * $x;
    } else if (!isset($z)) {
      return $x * $y;
    } else {
      return $x + $y + $z;
    }
  }

  // Ejemplo de sobrecarga de un método según el
  // tipo de los parámetros que se pasan.
  //
  // Si se pasan dos números enteros se devuelve
  // la suma; en caso contrario se muestran los
  // parametros separados por coma.
  function opera2($a, $b) {
    if (is_int($a) && is_int($b)) {
      return $a + $b;
```

```
    } else {
      return $a . ", " . $b;
    }
  }
```

La función `opera()` se comporta como tres funciones en una sola. Si se le pasa un único número, la función devuelve el cuadrado de ese número; si se pasan dos parámetros, devuelve la multiplicación y si se pasan tres, devuelve la suma de todos ellos. Para comprobar si se pasa o no un determinado parámetro se utiliza la función `isset()`.

La función `opera2()` se comporta como dos funciones en una. Esta vez la simulación de la sobrecarga no se realiza en función del número de parámetros sino del tipo. Mediante `is_int()` podemos comprobar si un determinado parámetro es un número entero. En el caso que nos ocupa, si la función `opera2()` recibe como parámetros dos números enteros, se devuelve la suma; en caso contrario, lo que se devuelve es una cadena de caracteres con los parámetros que se han pasado separados por coma.

El programa principal que llama a las funciones es el siguiente.

Observa que en ambos ficheros se especifica que las clases declaradas (y por tanto las funciones que se definen dentro) pertenecen al paquete `matematicas` mediante la línea `package matematicas`. Ahora ya podemos probar las funciones desde un programa externo. El programa `PruebaFunciones.java` está fuera de la carpeta `matematicas`, justo en un nivel superior en la estructura de directorios.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'funciones.php';

      // Mismo nombre de función con diferente
      // número de parámetros.
      echo opera(2)."<br>";
      echo opera(2, 3)."<br>";
      echo opera(2, 3, 10)."<br>";

      // Mismo nombre de función con distintos
      // tipos de parámetros.
      echo opera2(10, 20)."<br>";
      echo opera2("melón", "sandía")."<br>";
    ?>
```

```
  </body>
</html>
```

La salida del programa anterior es la siguiente:

```
4
6
15
30
melón, sandía
```