Desarrollo Web en entorno servidor

Unidad 2:
Introducción a PHP. Parte 3
Funciones y ámbitos de variables

Tabla de contenido

1. Funciones

La mayor parte de las secciones del código de un script PHP deben ser ejecutadas tan pronto como dicho script es interpretado, pero, otras veces, es preferible que el código actúe después de que se haya producido alguna acción específica o sólo si se dispara un evento.

También es habitual que partes del código se repitan un número indeterminado de veces durante la ejecución del script PHP. Estas necesidades hacen que nazca la idea de dividir el código de un script en partes menores, para que cada una de las cuales sirva a un propósito específico e individual.

Una función PHP es simplemente una sección separada de código a la que se le ha dado un nombre (cualquier instrucción PHP válida puede aparecer en el cuerpo de la función, incluso la llamada a otra función o la definición de clases). Utilizando este nombre, en un script se puede llamar a esta sección de código tantas veces como se quiera y en los momentos en que se necesite.

Por tanto, las funciones dividen las tareas que debe hacer un script, agrupando instrucciones relacionadas para la ejecución de una tarea. Esta estructuración del código nos permite escribir scripts más sencillos, legibles y fáciles de entender.

Las funciones pueden recibir valores desde las sentencias que las llaman. Estos valores se denominan parámetros o argumentos y pueden devolver valores. Los parámetros, como veremos más adelante, se usarán como variables locales dentro del bloque de sentencias que conforman la función.

1.1. Declaración de una función

La sintaxis de la declaración de una función es la siguiente:

```
function nombreFunción ([parámetrol [...]){

     Sentencia 1;
     Sentencia N;
}
```

La palabra reservada `function` se utiliza para especificar un nombre, `nombreFunción`, el cual sirve como identificador para el conjunto de sentencias comprendidas entre las llaves. Encerrados entre paréntesis y separados por comas, se encuentran los nombres de los parámetros, que son los que recibirán los valores con los que es llamada la función.

Técnicamente, los argumentos son variables que contienen informaciones necesarias para que la función realice correctamente labor y que son pasados a ella en la sentencia de llamada. Aunque no se incluyan parámetros, en la declaración de la función deben escribirse los paréntesis. Las sentencias, que conforman el núcleo de la función, son ejecutadas cada vez que se llama a la función.

Las funciones pueden devolver un valor usando la sentencia `return`. Cuando en una función se encuentra una sentencia `return`, termina su procesamiento y devuelve el valor que se indica.

Hasta PHP3 las funciones debían definirse antes de ser llamadas Desde PHP4 no existe esta restricción.

1.2. Paso de argumentos a una función

PHP permite pasar los parámetros de tres formas distintas por valor, por referencia y con parámetros por defecto.

1.2.1. Parámetros por valor

En el caso del paso de parámetros por valor, que es la opción por defecto en PHP, lo que recibe la función es una copia del valor de la variable pasada como parámetro de esta forma, las modificaciones que puedan hacerse dentro del cuerpo de la función a la variable parámetro no afectan al valor final de la variable pasada como argumento.

1.2.2. Parámetros por referencia

En el caso de que queramos que los cambios que se producen en el cuerpo de la función afecten a la variable que se pasó como argumento en la llamada a la función deberemos pasar el parámetro por referencia. Como su propio nombre indica, en este caso, a la función le llega una referencia a la variable y, por tanto, los cambios que realice sobre el parámetro se realizan sobre la variable.

Para indicar qué parámetros se pasan por referencia, hay que marcarlos en la definición de la función, anteponiendo el símbolo ampersand (&) al nombre del parámetro.

1.2.3. Parámetros por defecto

Los parámetros por defecto son la forma en que PHP implementa los parámetros opcionales en la llamada a las funciones. De este modo, este tipo de parámetros toma un valor predefinido cuando, desde la llamada a la función, no se les ha proporcionado ningún argumento. Para definir un parámetro por omisión, hay que, además de nombrar el parámetro, escribir el operador de asignación "=", a continuación, el valor que vaya a recibir el parámetro en caso de no especificarse en la llamada.

Cuando se usan parámetros por defecto, éstos tienen que situarse los últimos en la declaración, es decir, a la derecha de cualquier parámetro normal.

El siguiente ejemplo, `parametros.php` muestra el uso de los tres tipos de parámetros.

```
 1:  <HTML>
 2:
 3:  <HEAD>
 4:      <TITLE>Trabajando con Funciones</TITLE>
 5:  </HEAD>
 6:
 7:  <BODY>
 8:      <H2>Funciones de Usuario</H2>
 9:      <?php
10:      $mifinal = 0;
11:      function cuentaAtras($inicio, &$fin, $mensaje = "¡ Y cero !")
12:      {
13:          for ($i = $inicio; $i > $fin; $i--) {
14:              echo $i, "...<br>";
15:          }
16:
```

```
17:           $fin = $fin + 2;
18:           echo $mensaje;
19:       }
20:       ?>
21:       <TABLE>
22:           <tr>
23:               <td>
24:                   <?php
25:                   // $mifinal vale 0
26:                   cuentaAtras(6, $mifinal);
27:                   // $mifinal vale 2
28:                   ?>
29:               </td>
30:               <td>
31:                   <?php
32:                   // $mifinal vale 2
33:                   cuentaAtras(8, $mifinal, "¡ Terminé en " . $mifinal . " !");
34:                   // $mifinal vale 4
35:                   ?>
36:               </td>
37:       </TABLE>
38:   </BODY>
39:
40:   </HTML>
```

En el script `duplicar.php` vemos un ejemplo del uso de `return` para devolver el valor adecuado al duplicar una cantidad, haciendo uso del paso por valor frente al no uso de `return` (`duplicarMal()`) y al paso por referencia (`duplicar2()`).

```
 1:  <?php
 2:      function duplicarMal($a){
 3:          $a = $a *2;
 4:      }
 5:      function duplicar($a){
 6:          return $a *2;
 7:      }
 8:      function duplicar2(&$a){
 9:          $a = $a *2;
10:      }
11:      $var1 = 5;
12:      duplicarMal($var1);
13:      echo "$var1 <br>";
14:      $var1 = duplicar($var1);
15:      echo "$var1 <br>";
16:      duplicar2($var1);
17:      echo "$var1 <br>";
```

1.3. Llamada a una función desde ficheros externos.

Conforme van creciendo los programas resulta trabajoso encontrar la información que buscas dentro del código. En ocasiones resulta útil agrupar ciertos grupos de funciones o bloques de código, y ponerlos en un fichero aparte. Posteriormente, puedes hacer referencia a esos ficheros para que PHP incluya su contenido como parte del programa actual.

Para incorporar al programa contenido de un archivo externo, existen varias posibilidades:

`include`: Evalúa el contenido del fichero que se indica y lo incluye como parte del fichero actual, en el mismo punto en que se realiza la llamada. La ubicación del fichero puede especificarse utilizando una ruta absoluta, pero lo más usual es con una ruta relativa. En este caso, se toma como base la ruta que se especifica en la directiva `include_path` del fichero `php.ini`. (C:\xampp\php) Si no se encuentra en esa ubicación, se buscará también en el directorio del guion actual, y en el directorio de ejecución.

Usamos los ejemplos: `definiciones.inc.php` y `programa.php`

`Definicions.inc.php`

```
1:  <?php
2:      $modulo = 'DWES';
3:      $ciclo = 'DAW';
4:  ?>
```

`programa.php`

```
1:  <?php
2:      print "Módulo $modulo del ciclo $ciclo<br />"; //Solo muestra
"Modulo del ciclo"
3:      include 'definiciones.php';
4:      print " Módulo $modulo del ciclo $ciclo<br />"; // muestra "Modulo
DWES del ciclo DAW"
5:  ?>
```

Cuando se comienza a evaluar el contenido del fichero externo, se abandona de forma automática el modo PHP y su contenido se trata en principio como etiquetas HTML. Por este motivo, es necesario delimitar el código PHP que contenga nuestro archivo externo utilizando dentro del mismo los delimitadores <?php y ?>.

`include_once`: Si por equivocación se incluye más de una vez un mismo fichero, lo normal es obtener algún tipo de error (por ejemplo, al repetir una definición de una función). `include_once` funciona exactamente igual que `include`, pero solo incluye aquellos ficheros que aún no se hayan incluido.

`require`: Si el fichero que queremos incluir no se encuentra, `include` da un aviso y continua la ejecución del guion. La diferencia más importante al usar `require` es que en ese caso, cuando no se puede incluir el fichero, se detiene la ejecución del guion.

`require_once:` Es la combinación de las dos anteriores. Asegura la inclusión del fichero indicado solo una vez, y genera un error si no se puede llevar a cabo.

Es práctica habitual utilizar la doble extensión `.inc.php` para aquellos ficheros en lenguaje PHP cuyo destino es ser incluidos dentro de otros, y nunca han de ejecutarse por sí mismos.

1.4. Funciones como argumentos. Callbacks

En PHP es posible pasar funciones como argumentos a otras funciones. Es una característica avanzada que se utiliza entre otras cosas para las funciones de callback. Solo hay que pasar el nombre de la función entre comillas como argumento. La función que lo recibe podrá utilizarla si se le pasan los argumentos adecuados. En el ejemplo `calculador.php` se utiliza esta característica. La función `calculador()` recibe como argumentos dos números y también la función que debe aplicarles. Según qué función se le pase como argumento, devolverá un valor u otro.

```
 1:  <?php
 2:      function calculador($operacion, $numa, $numb){
 3:          $resul = $operacion($numa, $numb);
 4:          return $resul;
 5:      }
 6:      function sumar($a, $b){
 7:          return $a + $b;
 8:      }
 9:      function multiplicar($a, $b){
10:          return $a * $b;
11:      }
12:      $a = 4;
13:      $b = 5;
14:      $r1 = calculador("multiplicar", $a, $b);
15:      echo "$r1 <br>";
16:      $r2 = calculador("sumar", $a, $b);
17:      echo "$r2 <br>";
```

1.5. Recursividad

Se dice que una función es recursiva cuando en algún punto de su cuerpo se llama a sí misma. Hay que tener cuidado al escribir una función recursiva, ya que puede ocurrir que se llame a sí misma indefinidamente. Por tanto, es esencial asegurarse de implementar una forma adecuada de terminar la recursión: es lo que se denomina como condición de parada.

Veamos un ejemplo para hallar el factorial de un número mediante una función recursiva: `factorial_recursivo.php`

```
1:  <?php
2:  $numero = 5;
3:  function factorial ($numero) {
4:      if ($numero==0) return 1;
5:      return $numero* factorial ($numero-1);
6:  }
7:  echo "El factorial de $numero es ". factorial(5);
```

1.6. Funciones predefinidas

PHP cuenta con una gran cantidad de funciones predefinidas para las tareas más habituales. Entre las más utilizadas están las relacionadas con las variables. Como en PHP las variables no se declaran explícitamente, hay una serie de funciones que permiten conocer el estado y tipo de dato de una variable. Las más importantes son:

- `is_null($var)`. Devuelve TRUE si `$var` es NULL, FALSE en otro caso.
- `isset($var)`. Devuelve TRUE si `$var` ha sido inicializada y su valor no es NUL FALSE en otro caso.
- `unset ($var)`. Elimina la variable. Ya no contará como inicializada. `empty($var)`. Devuelve TRUE si `$var` no ha sido inicializada o su valor es FALSE FALSE en otro caso.
- `is_int ($var)`, `is_float ($var)`, `is_bool ($var)`, `is_array($var)`. Devuelven TRUE si `$var` es entero, float, booleano o array respectivamente, y FALSE en otro caso.
- `print_r($var)` y `var_dump($var)`. Muestran información sobre $var.

En el ejemplo `funciones_variables.php` se pueden ver las diferencias entre estas funciones

```
 1:  <?php
 2:     $var1 = 4;
 3:     $var2 = NULL;
 4:     $var3 = FALSE;
 5:     $var4 = 0;
 6:     echo " var 1 = 4 <br>· ¿inicializada? ";
 7:     var_dump(isset($var1));     // TRUE
 8:     echo " <br>· ¿es nula? ";
 9:     var_dump(is_null($var1));  // FALSE
10:     echo " <br>· ¿es falso o vacia? ";
11:     var_dump(empty($var1));     // FALSE
12:     echo "<br> var 2 = NULL <br>· ¿inicializada? ";
13:     var_dump(isset($var2));     // FALSE
14:     echo " <br>· ¿es nula? ";
15:     var_dump(is_null($var2));  // TRUE
16:     echo " <br>· ¿es falso o vacia? ";
17:     var_dump(empty($var2));     // TRUE
18:     echo "<br> var 3 = FALSE <br>· ¿inicializada? ";
19:     var_dump(isset($var3));     // TRUE
20:     echo " <br>· ¿es nula? ";
21:     var_dump(is_null($var3));  // FALSE
22:     echo " <br>· ¿ss falso o vacia? ";
23:     var_dump(empty($var3));     // TRUE
24:     echo "<br> var 4 empty <br>· ¿es falso o vacia? ";
25:     var_dump(empty($var4));     // TRUE, EL 0 COMO BOOLEAN ES FALSE
26:     echo "<br> var 1 unset <br>· ¿Inicializada? ";
27:     unset($var1);
28:     var_dump(isset($var1));     // FALSE
```

2. Ámbito de las variables y variables predefinidas.

El ámbito de una variable es la parte del código en que esta es visible. Una variable declarada en un fichero PHP está disponible en ese fichero y en los ficheros que se incluyan desde este.

Por otro lado, las funciones definen un ámbito local, de manera que las variables que se declaran en las mismas solo son accesibles desde la propia función. Además, desde la función no se puede acceder a otras variables que no sean las locales o sus argumentos.

Para definir variables globales hay dos opciones. La palabra reservada global y la variable predefinida $_GLOBALS, la cual es un array asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del array. Las variables globales son accesibles desde cualquier función, o fichero de la aplicación.

Por tanto, respecto a su ámbito, podemos definir dos tipos de variables: globales y locales. Si queremos que una variable local no pierda su valor tras abandonar el ámbito de la misma en la ejecución del programa, la declararemos como `static`

Veamos el ejemplo `ambitos.php`

```
 1:  <HTML>
 2:  <HEAD>
 3:      <TITLE>Trabajando con Funciones</TITLE>
 4:  </HEAD>
 5:  <BODY>
 6:      <H2>Funciones de Usuario</H2>
 7:      <?php
 8:      $inicio = 9;
 9:      $final = 0;
10:      function cuentaAtras(){
11:          // variable global
12:          global $final;
13:          // variable local
14:          $inicio = 7;
15:          // variable estática
16:          static $num = 0;
17:          for ($i = $inicio; $i > $final; $i--) {
18:              echo $i, "... <br>";
19:          }
20:          $num++;
21:          echo "¡ FIN -$num- !";
22:      }
23:      ?>
24:      <TABLE>
25:          <td>
26:              <?php
27:              cuentaAtras(); // $num vale 1
28:              ?> </td>
29:          <td>
30:              <?php
```

```
31:                 cuentaAtras(); // $num vale 2
32:                 ?>
33:             </td>
34:             </TR>
35:         </TABLE>
36:     </BODY>
37:     </HTML>
```

2.1. Variables predefinidas

En PHP hay muchas variables predefinidas disponibles. Contienen información sobre el servidor, datos enviados por el cliente o variables de entorno. Dentro de las variables predefinidas hay un grupo de ellas, las superglobales, que están disponibles en cualquier ámbito.

Cada una de ellas guarda información de un tipo.

Por ejemplo, en $_SERVER hay información sobre el servidor en el que está alojada la página. El scrip `global_server.php` muestra algunos de los datos disponibles.

```
1:  <?php
2:   echo "Ruta dentro de htdocs: " . $_SERVER['PHP_SELF']. "<br>";
3:   echo "Nombre del servidor: " . $_SERVER['SERVER_NAME']. "<br>";
4:   echo "Software del servidor: " . $_SERVER['SERVER_SOFTWARE']. "<br>";
5:   echo "Protocolo: " . $_SERVER['SERVER_PROTOCOL']. "<br>";
6:   echo "Método de la petición: " . $_SERVER['REQUEST_METHOD']. "<br>";
```

Las variables superglobales son muy relevantes para el desarrollo de aplicaciones web y se irán poniendo en práctica a lo largo del curso.

NOMBRE	DESCRIPCIÓN
$GLOBALS	Variables globales definidas en la aplicación
$_SERVER	Información sobre el servidor
$_GET	Parámetros enviados con el método GET (en la URL)
$_POST	Parámetros enviados con el método POST (formularios)
$_FILES	Ficheros subidos al servidor
$_COOKIE	Cookies enviadas por el cliente
$_SESSION	Información de sesión
$_REQUEST	Contiene la información de $_GET, $_POST y $_COOKIE
$_ENV	Variables de entorno

3. Actividades obligatorias

Actividad obligatoria Nº 4

Crea una función en un archivo con extensión .inc.php que permita calcular el factorial de un número mediante un bucle for. El número se pasará por valor a dicha función.

Crear otro fichero con extensión .php que contenga una función que permita calcular el número combinatorio C(m,n), a partir de las variable $m y $n previamente inicializadas a los valores 5 y 3, respectivamente, donde:

$$C(m,n) = \binom{n}{m} = \frac{n!}{m!\,(n-m)!}$$

Ten en cuenta que $0! = 1$; y que si $n < m$ o $n \leq 0$, entonces el número combinatorio no existe.

La función tiene que mostrar en HTML "El número combinatorio C($m,$n) no existe" si fuera el caso, o bien "El número combinatorio C($m,$n) vale $resultado".

Actividad obligatoria Nº 5

La conjetura de Collatz afirma que si tomamos cualquier número natural y le aplicamos iterativamente la siguiente operación

- **Dividimos entre dos si el número es par.**
- **Multiplicamos por 3 y sumamos 1 si el número es impar.**

el resultado, tras un número determinado de pasos, será siempre 1.

Ejemplo:

Si empezamos por el número 10, dividimos entre 2 al ser 10 un número par y obtenemos 5.

Ahora multiplicamos a 5 por 3 y le sumamos 1, obteniendo 16 (5*3+1).

Como 16 es par, dividimos entre 2 y obtenemos 8

Como 8 es par, dividimos entre 2 y obtenemos 4, dividimos de nuevo y obtenemos 2 y por último 1.

El número de pasos que hemos realizado han sido:

La secuencia de números obtenidos para el 10 son: 5, 16, 8, 4, 2, 1.

Escribe un script que contenga una función recursiva que calcule la secuencia de números. El HTML ha de mostrar el número de pasos realizados y los números obtenidos en cada paso.