ÍNDICE

3 CONCEPTOS AVANZADOS..... 1

3.1 Cadenas.....1
3.1.1 Combinar código HTML con código PHP.....1
3.1.2 Manipulación de cadenas.....5
3.2 Arrays.....9
3.2.1 Arrays indexados numéricamente.....9
3.2.2 Arrays no secuenciales.....12
3.2.3 Arrays asociativos.....13
3.2.4 Arrays no homogéneos.....14
3.2.5 Arrays multidimensionales.....14
3.2.6 Mostrar el contenido de un array.....15
3.2.7 Desbordamiento de arrays.....15
3.2.8 Sentencia foreach.....16
3.3 Funciones.....16
3.3.1 Funciones definidas por el usuario.....17
3.3.2 Creación de bibliotecas.....21
3.3.3 Otros tipos de funciones.....22
3.4 Formularios.....24
3.4.1 Formularios HTML.....24
3.4.2 Recepción de datos de formularios.....28
3.5 Manejo de ficheros.....33
3.5.1 Manejo de ficheros.....33
3.5.2 Atributos de un fichero.....36
3.5.3 Manejo de directorios.....37
3.5.4 Subir archivos al servidor.....38

3 CONCEPTOS AVANZADOS

3.1 Cadenas.

El manejo de cadenas en *PHP* es de vital importancia, ya que la única forma que tenemos de comunicarnos con el cliente es a través de texto.

Además, hay que tener en cuenta que el producto de un programa en *PHP* es una página web con formato *HTML*. Es decir, un fichero de texto.

3.1.1 Combinar código HTML con código PHP.

Como en una página web dinámica programada en *PHP* hay código de dos tipos, lo mejor es mantenerlos bien diferenciados. Hay que ser especialmente cuidadoso, procurando que el código sea muy legible. Esto nos facilitará el posterior mantenimiento de la página, y hará mucho más sencillo dotarla de elementos de diseño.

El objetivo ideal sería que un diseñador web pudiera aplicar diseño a una página *PHP*, sin tener que preocuparse de la programación que contiene. En niveles más avanzados se utilizan sistemas de plantillas que ayudan notablemente en la tarea de separar el código *HTML* y *PHP*.

3.1.1.1 Variables dentro de cadenas.

Hasta ahora, siempre que hemos necesitado mostrar un texto compuesto por una cadena y el valor de una variable los hemos tenido que unir usando el operador concatenar, ".". Pero *PHP* ofrece una sintaxis alternativa, que resulta más compacta y puede ser de utilidad en muchos casos.

Ejemplo 3.1.1. Si ponemos una variable en el interior de una cadena entre comillas, *PHP* la interpreta, y en lugar del nombre de la variable escribe su valor. Lo podemos comprobar a continuación, donde los dos fragmentos de código producen los mismos resultados.

```php
<?php
  $edad = 46;
  echo "Belinda tiene " . $edad . " años.";
?>

<?php
  $edad = 46;
  echo "Belinda tiene $edad años.";
?>
```

3.1.1.2 Uso de la sentencia echo.

Es recomendable usar esta forma de producir la salida cuando es predominante la cantidad de código *PHP* sobre la cantidad de código *HTML*. Lo mismo se aplica al uso de la función *print()* que produce los mismos resultados.

Hay partidarios de usar una forma y partidarios de usar la otra. Independientemente de cual se use, no es muy recomendable mezclarlas dentro del mismo programa.

Ejemplo 3.1.2. En este pequeño programa predomina el código *PHP* sobre el código *HTML*.

```php
<html>
  <head>
    <title>Números primos</title>
  </head>
  <body>
<?php
  define("LIMITE", 100);
  echo "Los números primos del 1 al " . LIMITE . " son:";
  echo "<br>";
  for ($i = 1; $i <= LIMITE; $i++) {
    $es_primo = TRUE; //Hipótesis inicial, es primo
    for ($j =2; $j < $i; $j++) {
      if(($i % $j) == 0) { //Tiene un divisor
        $es_primo = FALSE; //Luego no es primo
        break;
      }
    }
    if ($es_primo) {
      echo "$i";
      echo "<br>";
    }
  }
?>
  </body>
</head>
```

3.1.1.3 Uso de la sintaxis <?= ... ?>.

A la inversa, cuando predomina el código *HTML* sobre el código *PHP* puede ser interesante mostrar la salida usando esta sintaxis. De cualquier modo, debemos tener presente que esta etiqueta no está disponible en todos los servidores web, y podemos encontrarnos algunos casos en

que, debido a la configuración del propio servidor, esta etiqueta no funcione adecuadamente.

Ejemplo 3.1.3. En este pequeño programa predomina el código *HTML* sobre el código *PHP*.

```
<html>
  <head>
    <title>El hombre del tiempo</title>
  </head>
<body bgcolor="#cccccc">
  <table border="1">
    <tr>
      <td align="center" colspan="2">
        Hola, soy el hombre del tiempo.<br>
        Mi método para adivinar el clima es infalible.
      </td>
    </tr>
    <tr>
      <td align="center">
        Temperatura mínima prevista
      </td>
      <td align="center">
        Temperatura máxima prevista
      </td>
    </tr>
    <tr>
      <td align="center">
        <font size="+2"><?= rand(0, 15)?></font> grados.
      </td>
      <td align="center">
        <font size="+2"><?= rand(15, 30)?></font> grados.
      </td>
    </tr>
  </table>
</html>
```

3.1.1.4 Fragmentos de HTML dentro de estructuras condicionales.

Cuando tenemos que mostrar trozos de código *HTML* muy grandes, y éste se encuentra dentro de una estructura condicional, puede ser muy pesado construirlos a base de sentencias *echo*.

En este caso es preferible volver a cerrar el bloque de código *PHP* con la etiqueta “?>”, escribir el código *HTML* que se ejecuta condicionalmente, y abrir otra vez el código *PHP* con la etiqueta “<?php” para terminar la estructura condicional.

De este modo, reducimos la mezcla e interferencia entre etiquetas y sentencias de ambos lenguajes, lo que repercute en una mayor claridad y sencillez. Además, este sistema combinado con el uso de la sintaxis “<=? ... ?>” produce un código compacto y fácil de leer.

Ejemplo 3.1.4. Aquí se ven las dos estrategias. La segunda es más clara y comprensible.

```php
<?php
  $autorizado = FALSE;
  if (!$autorizado) {
    echo '<center>';
    echo '<table bgcolor="#cccccc">';
    echo '<tr>';
    echo '<td align="center">';
    echo '<font color="#ff0000" size="+2">';
    echo 'No está autorizado a ver esta página.';
    echo '<br>';
    echo 'Contacte con el administrador del sistema.';
    echo '</font>';
    echo '</td>';
    echo '</tr>';
    echo '</table>';
    echo '</center>';
  }
  echo '<br>';
?>

<?php
  $autorizado = FALSE;
  if (!$autorizado) {
?>
<center>
  <table bgcolor="#cccccc">
    <tr>
      <td align="center">
        <font color="#ff0000" size="+2">
          No está autorizado a ver esta página.<br>
          Contacte con el administrador del sistema.
        </font>
      </td>
    </tr>
  </table>
</center>
<?php
  }
?>
```

3.1.1.5 Distribución del código.

La última cuestión que nos queda por resolver es donde debemos situar el código. Salvo el código que muestra resultados por pantalla, que tiene que aparecer en el lugar apropiado de la página, el código *PHP* se puede situar en cualquier lugar del fichero. La mejor opción es separar en la medida de lo posible el código encargado de:

- Inicializar las constantes y variables.
- Leer datos de entrada.
- Realizar operaciones.

Este código es conveniente ponerlo al principio de la página. Los resultados de las operaciones que se quieran mostrar se almacenan entonces en variables auxiliares y se muestran en el cuerpo de la página HTML donde corresponda haciendo uso de las etiquetas “<?= ... ?>”.

Ejemplo 3.1.5. Esta es una página web en *PHP* bien construida. Usa un algoritmo para hallar el máximo común divisor de dos números generados aleatoriamente.

```
<?php
  // Inicialización
  define("LIMITE", 100);
  // Lectura de datos
  $i = rand(1, LIMITE); // Simulamos la lectura
  $j = rand(1, LIMITE);
  // Operaciones con los datos
  if ($i > $j) {
    $mayor = $i;
    $menor = $j;
  } else {
    $mayor = $j;
    $menor = $i;
  }
  while (($mayor % $menor) != 0) {
    $auxiliar = $menor;
    $menor = $mayor % $menor;
    $mayor = $auxiliar;
  }
  $mcd = $menor;
?>
<html>
  <head>
    <title>Máximo comun divisor</title>
  </head>
  <body>
    El máximo común divisor de <?= $i ?> y <?= $j ?> es <?= $mcd ?>.
  </body>
</html>
```

Esto no es posible siempre. Por ejemplo, si tenemos que mostrar una lista de 11 resultados el bucle que los recorre deberá estar mezclado con el código *HMTL*. De todas formas con la experiencia iremos aprendiendo poco a poco a extraer la mayor parte del código *PHP* al comienzo de la página.

3.1.2 Manipulación de cadenas.

PHP dispone de funciones predefinidas en el núcleo que nos permiten hacer operaciones con los textos almacenados en cadenas. Ya hemos visto las función *echo* y *printf()* pero hay unas cuantas más.

3.1.2.1 Acceder a los caracteres de una cadena.

Se puede acceder a los carácteres de las cadenas indicando la posición del carácter que queremos extraer. Las posiciones van desde 0 hasta el número de caracteres (longitud) menos uno. Si intentamos acceder a una posición más allá del final de la cadena nos devolverá la cadena vacía (cadena sin ningún carácter = "").

Ejemplo 3.1.6. Se muestran varias pruebas de extracción de caracteres de una cadena.

```php
<?php
  $cadena = "Pernambuco";
  echo "Cadena a analizar: '$cadena'<br>";

  // Caracter en la primera posición
  echo "Caracter en la posición 0: '$cadena[0]'<br>";

  // Caracteres de posiciones intermedias
  echo "Caracter en la posición 1: '$cadena[1]'<br>";
  echo "Caracter en la posición 5: '$cadena[5]'<br>";
  echo "Caracter en la posición 7: '$cadena[7]'<br>";

  // La cadena tiene longitud 10, la última posición es la 9
  echo "Caracter en la posición 9: '$cadena[9]'<br>";

  // La cadena tiene longitud 10, en la posición 12 no hay nada
  echo "Caracter en la posición 12: '$cadena[12]'<br>";
?>
```

3.1.2.2 strlen().

Por lo general no conoceremos la longitud de la cadena a priori. Esta función nos permite obtenerla.

```
strlen(cadena)
```

3.1.2.3 strpos().

En ocasiones nos interesará conocer la posición de la primera aparición de un carácter en una cadena. Si no se encuentra el carácter, devuelve FALSE.

```
strpos(cadena, caracter)
```

Alternativamente podemos indicarle la posición a partir de la cual queremos que empiece a buscar.

```
strpos(cadena, caracter, posición inicial)
```

3.1.2.4 substr().

Habrá muchos casos en los que nos interesará extraer un trozo del texto de una cadena. Mediante esta función podemos extraer el fragmento ubicado entre las posiciones indicadas:

```
substr(cadena, posición inicial, posición final)
```

Ejemplo 3.1.7. El funcionamiento de esta función se explicará con un ejemplo típico, la comprobación de que un mail es correcto y la extracción de datos del mismo.

```php
<?php
  $email = "maurodosantos@pernambuco.com ";
  echo "Email a analizar: '$email'<br>";
  echo "<br>";
  echo "Tiene " . strlen($email) . " letras.<br>";
  // Indica la posición del caracter "@" o FALSE si no está
  $posicion_arroba = strpos($email, "@");
  // Busca la aparición de un punto (.) partir de la arroba
  $posicion_punto = strpos($email, ".", $posicion_arroba);
  if ($posicion_arroba && $posicion_punto) {
    echo "Es una dirección de email válida<br>";
    $usuario = substr($email, 0, $posicion_arroba);
    $dominio = substr($email, $posicion_arroba + 1);
    echo "El nombre de usuario es: $usuario<br>";
    echo "El dominio es: $dominio<br>";
  } else {
    echo "No es una dirección de email válida<br>";
    if (!$posicion_arroba) {
      echo "Le falta el caracter arroba<br>";
    }
    if (!$posicion_punto) {
      echo "El dominio no es válido<br>";
    }
  }
?>
```

3.1.2.5 trim().

Elimina los espacios en blanco al principio y final de una cadena.

```
trim(cadena)
```

3.1.2.6 str_replace().

En un texto, reemplaza las apariciones de una cadena por otra.

```
str_replace(cadena a buscar, cadena reemplazar, cadena original)
```

Ejemplo 3.1.8. En este ejemplo se supone que podemos recibir emails con errores, y los intentamos arreglar. El primer error que eliminamos es la aparición de espacios en blanco al principio y al final de la dirección. El segundo cuando los usuarios escriben el dominio *pernambuco.es* cuando el correcto es *pernambuco.com*.

```php
<?php
  $email = " maurodosantos@pernambuco.es ";
  echo "Dirección recibida: '$email'.<br>";
  // Eliminamos los espacios en blanco
  $email = trim($email);
  // Reemplazamos pernambuco.es por pernambuco.com
  $email = str_replace("pernambuco.es", "pernambuco.com", $email);
  echo "Dirección corregida: '$email'.";
?>
```

3.1.2.7 strtoupper() y strtolower().

La función *strtoupper()* convierte los textos a mayúsculas, y *strtolower()* a minúsculas.

```
strtoupper(cadena)
strtolower(cadena)
```

Ejemplo 3.1.9. En este ejemplo vamos a pasar la dirección de correo a minúsculas.

```php
<?php
  $email = " MAUROSOSANTOS@PERNAMBUCO.ES ";
  echo "Dirección recibida: '$email'.<br>";

  // Convertimos a minúsculas
  $email = strtolower($email);
  echo "Dirección corregida: '$email'.";
?>
```

Ejemplo 3.1.10. Un uso muy habitual de estas funciones es para comparar cadenas.

Para *PHP* la cadena "MADRID" es diferente a la cadena "Madrid", pero mediante estas funciones se pueden comparar como iguales.

```php
<?php
  echo "Comparación de cadenas directamente: ";
  if ("MADRID" == "Madrid") {
    echo "MADRID es igual a Madrid<br>";
  } else {
    echo "MADRID no es igual a Madrid<br>";
  }
  echo "<br>";

  echo "Antes de comparar pasamos ambas a minúsculas: ";
  if (strtolower("MADRID") == strtolower("Madrid")) {
    echo "MADRID es igual a Madrid<br>";
  } else {
    echo "MADRID no es igual a Madrid<br>";
  }
?>
```

3.1.2.8 number_format().

Esta función es útil cuando hay que mostrar datos de tipo double, es decir, números con decimales. Permite especificar el número de decimales que queremos que se muestren, en lugar de los diez decimales que se muestran habitualmente.

```
number_format(número, decimales)
```

Ejemplo 3.1.11. Esta función ya apareció en una práctica del módulo anterior.

```php
<?php
  $precio_kg = 1.29;
  $peso = 2.17;
  $a_pagar = $precio_kg * $peso;
  // Saca el dato $a_pagar ocn 2 decimales
  echo "A pagar " . number_format($a_pagar, 2) . " euros.";
?>
```

3.2 Arrays.

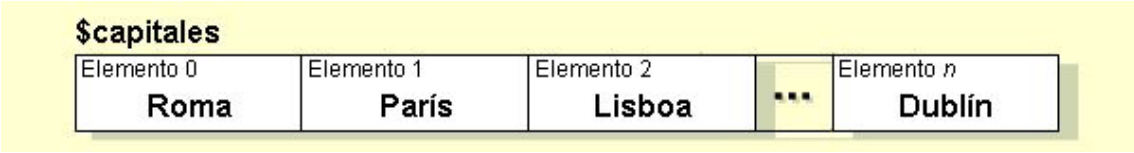
Hasta ahora hemos trabajado en nuestros programas con una pequeña cantidad de datos, y sin que la cantidad de los mismos variara a lo largo del programa. Sin embargo, la mayoría de los programas útiles requieren tratar gran cantidad de datos. En este caso sería muy incómodo tener una variable para cada dato.

Afortunadamente, cuando se tratan muchos datos, éstos suelen ser de unos tipos muy similares entre sí. Para tratarlos en grupo, *PHP* ofrece tipos de datos compuestos. Es decir, tipos de datos que permiten almacenar varios datos en una misma variable.

El tipo de datos compuesto más sencillo es el *array*. Es una estructura muy potente, flexible y de uso muy intuitivo. Un array está compuesto por varios *elementos*. Cada *elemento* almacena un *valor* diferente. Y para poder localizar un *elemento* disponemos del *índice* o posición, que podría interpretarse como la dirección del dato en cuestión.

3.2.1 Arrays indexados numéricamente.

El tipo más sencillo de arrays son los indexados numéricamente, en los que el índice de cada elemento corresponde con su posición. Hay que tener en cuenta que, al igual que sucedía con las cadenas, el primer elemento de un array tiene *índice* 0, y no 1 como cabría esperar. De igual modo, el último elemento corresponde al *indice* longitud-1.



3.2.1.1 Inicializar un array.

Los arrays se suelen almacenar en variables, como cualquier otro tipo de datos.

La forma más sencilla de crear un array es asignando los valores de sus elementos a la vez que se crea. Para ello primero declaramos el nombre de la variable y usamos la función *array()*, a la que le pasamos como parámetros un grupo de valores separados por comas.

Ejemplo 3.2.1. Creación de un array por asignación directa mediante la función *array()*:

```php
<?php
  $edades = array(28, 43, 32, 55);
  $formas = array("triángulo", "cuadrado", "círculo");
?>
```

La segunda forma de rellenar un array es añadiéndole elementos al final del array. Para añadir un elemento a un array se usa su identificador seguido de corchetes "[ ]" sin índice, y se le asigna un valor.

Ejemplo 3.3.2. Aquí se puede ver como se crea un array mediante la adición de elementos.

```php
<?php
  $paises[] = "Italia"; //Añade el elemento con índice 0
  $paises[] = "Francia"; //Añade el elemento con índice 1
  $paises[] = "Portugal"; //Añade el elemento con índice 2
?>
```

Ejemplo 3.3.3. Se pueden combinar ambas formas, primero declarar un array directamente y cuando sea necesario ir añadiendo elementos.

```php
<?php
  $frutas = array("melón", "sandía", "naranja");
  $frutas[] = "manzana";
  $frutas[] = "melocoton";
?>
```

3.2.1.2 Acceder a un elemento de un array.

Necesitaremos acceder a los elementos de un array con el fin de asignarles valores o de leer su contenido. Esto es muy sencillo, basta poner el índice del elemento al que queremos acceder entre corchetes "[...]".

Ejemplo 3.3.4. Aquí se leen los valores de los elementos de un array para sacarlos por pantalla. Luego se asigna un nuevo valor al primer elemento (recordamos, con índice 0).

```php
<?php
  $frutas = array("melón", "sandía", "naranja");
  echo "La primera fruta de la lista es $frutas[0].<br>";
  echo "La segunda fruta de la lista es $frutas[1].<br>";
  echo "La tercera fruta de la lista es $frutas[2].<br>";
  $frutas[0] = "limón"; //Asigna un nuevo valor al elemento 1
  echo "La primera fruta de la lista es $frutas[0].";
?>
```

3.2.1.3 Recorrer un array.

Los arrays se suelen utilizar para almacenar listas de valores. Por ello, una de las acciones más habituales que se hacen con ellos es recorrerlos de principio a fin para leer o modificar uno o varios de ellos.

Las estructuras iterativas (bucles) que se vieron con anterioridad son el mecanismo más apropiado para recorrer los arrays.

Para hacer dicho recorrido unicamente debemos limitar el número de iteraciones al número de elementos del array (longitud), que podemos averiguar mediante la función *count()*.

```
count(array)
```

Ejemplo 3.3.5. La función *count()* nos indica cuantos elementos contiene un array.

```php
<?php
  $frutas = array("melón", "sandía", "naranja");
  echo "La lista contiene " . count($frutas) . " frutas.";
?>
```

Ejemplo 3.3.6. Recorriendo un array para mostrarlo por pantalla.

```php
<?php
  $frutas = array("melón", "sandía", "naranja");
  for ($i = 0; $i < count($frutas); $i++) {
    echo "Elemento $i: $frutas[$i]<br>";
  }
?>
```

Ejemplo 3.3.7. Recorriendo un array para modificar todos sus elementos. Aquí se usan dos arrays que están relacionados por sus posiciones.

```php
<?php
  $nombres = array("Juan", "Inés", "Andrea", "Roberto");
  $edades = array(33, 28, 45, 52);
  for ($i = 0; $i < count($edades); $i++) {
    echo "$nombres[$i] tiene $edades[$i] años.<br>";
  }
  echo "<br>";

  for ($i = 0; $i < count($edades); $i++) {
    $edades[$i]++;
  }

  echo "Ha pasado un año...<br>";
  for ($i = 0; $i < count($edades); $i++) {
    echo "$nombres[$i] tiene $edades[$i] años. <br>";
  }
?>
```

Ejemplo 3.3.8. Recorriendo un array para buscar un elemento. Se vuelve a hacer uso de dos arrays relacionados por las posiciones de sus índices.

En este código se da por supuesto que se va a encontrar el valor buscado. Si no se encontrara el elemento buscado, el programa no daría respuesta alguna.

```php
<?php
  $nombres = array("Juan", "Inés", "Andrea", "Roberto");
  $edades = array(33, 28, 45, 52);
  echo "¿Cuantos años tiene Andrea?<br>";
  for ($i = 0; $i < count($nombres); $i++) {
    if ($nombres[$i] == "Andrea") {
      echo "$edades[$i] años";
    }
  }
?>
```

3.2.1.4 Ordenar un array.

La función *sort()* ordena los elementos de un array. Si el array está formado por cadenas de texto considera menores las minúsculas que las mayúsculas.

```
sort(array)
```

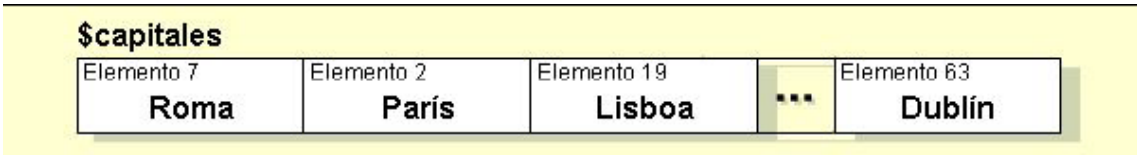
Ejemplo 3.3.9. En esta ejemplo se ordena una lista de nombres, primero por el procedimiento normal. Luego se muestra la forma de ordenar el mismo array sin hacer distinción entre mayúsculas y minúsculas.

```php
<?php
  $astros = array("planeta", "cometa", "Venus", "Jupiter");
  echo "Ordenación distinguiendo mayúsculas y minúsculas:<br>";
  sort($astros);
  for ($i = 0; $i < count($astros); $i++) {
    echo "$astros[$i]<br>";
  }
  echo "<br>";

  echo "Ordenación sin distinguir mayúsculas y minúsculas:<br>";
  $astros_minusculas = array_map("strtolower", $astros);
  array_multisort($astros_minusculas, SORT_ASC, SORT_STRING, $astros);
  for ($i = 0; $i < count($astros); $i++) {
    echo "$astros[$i]<br>";
  }
?>
```

3.2.2 Arrays no secuenciales.

Hasta ahora lo arrays que hemos visto tenían como índices la sucesión 0 (1er elemento), 1 (2o elemento), 2 (3er elemento), y así sucesivamente.

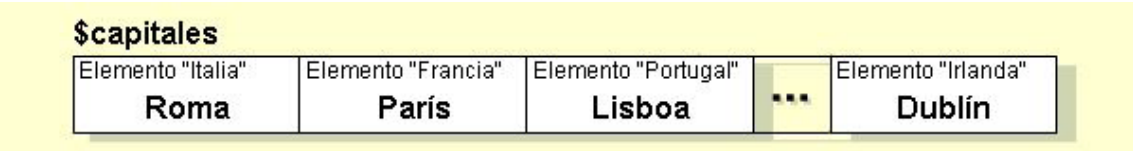


En *PHP* los índices de un array no tienen por qué ser consecutivos, pueden incluso estar desordenados.

El índice de los arrays asociativos no tiene por qué ser necesariamente un entero, puede ser también un número decimal o una cadena. Este tipo de arrays es más difícil de usar y, aunque permite una mayor flexibilidad, puede ser fuente de muchos errores difíciles de detectar si no se usa con cuidado.

3.2.3 Arrays asociativos.

Éstos son un caso específico de arrays no secuenciales, en los que el índice es una cadena de texto. Pueden resultar útiles para guardar listas, en las que se asocia un valor a una palabra clave.



La creación de este tipo de arrays se puede hacer de dos formas. La primera es mediante la función *array()* de forma parecida a como lo hacíamos antes, solo que ahora deberemos especificar un valor para el índice.

Ejemplo 3.2.10. Cuando se declaran explícitamente los índices se escribe el valor del índice seguido por "=>" y el valor del elemento que contiene.

```
<?php
  $capitales = array("Italia" => "Roma",
                     "Francia" => "Paris",
                     "Portugal" => "Lisboa");
  echo "La capital de Francia es {$capitales["Francia"]}";
?>
```

Como se puede observar en el ejemplo, si queremos hacer referencia a un elemento de un array indexado por una cadena dentro de un texto entre comillas, lo tendremos que encerrar entre llaves "{ }". En caso contrario, se produciría un error al interpretar la comilla de apertura del índice del array como si se tratase de la comilla final de la cadena.

Ejemplo 3.2.11. En la siguiente sentencia *PHP* considera como texto la parte en verde y como código la parte en rojo. Como no puede entender la palabra Francia dará un error.

```
echo "La capital de Francia es $capitales["Francia"]";
```

La segunda forma de crear un array asociativo es añadiendo elementos al array, y asignándoles de forma explícita cual es su índice.

Ejemplo 3.2.12. Veamos esta segunda forma de inicializar arrays declarando explícitamente sus índices.

```php
<?php
  $alturas["Aneto"] = 3404;
  $alturas["Teide"] = 3718;
  $alturas["Mulhacen"] = 3748;
  echo "El Aneto mide {$alturas["Aneto"]} metros";
?>
```

Hay que tener cuidado con este sistema de creación, pues se usa indistintamente para crear un elemento y para acceder a su valor.

Ejemplo 3.2.13. En este ejemplo primero se relaciona un array con otro a través de índices de tipo cadena. Los valores del primer array sirven a su vez como índices del segundo.

```php
<?php
  $nombres = array("Eva", "Antonio", "Felipe", "Rosa");
  $edades = array("Eva" => 28,
                  "Antonio" => 43,
                  "Felipe" => 32,
                  "Rosa" => 55);
  for ($i = 0; $i < count($nombres); $i++) {
    echo "$nombres[$i] tiene {$edades[$nombres[$i]]} años.<br>";
  }
?>
```

Aún así, al igual que ocurría con los arrays no secuenciales, su uso es desaconsejable si aún no se tiene cierto dominio sobre el lenguaje *PHP*.

3.2.4 Arrays no homogéneos.

PHP establece muy pocas limitaciones a las estructuras de los arrays. Aunque hasta el momento todos los arrays que hemos visto, los hemos usado para manipular datos del mismo tipo (arrays de enteros, arrays de cadenas, etc) esta restricción no existe realmente en *PHP*.

PHP permite mezclar en un array valores de diferentes tipos. Incluso permite que los índices de los elementos de un mismo array sean de diferentes tipos.

Pero una vez más, ésta es una estrategia peligrosa a seguir, y a la que en la mayoría de los casos no será necesario recurrir.

3.2.5 Arrays multidimensionales.

El contenido de un elemento de un array puede, a su vez, ser un array. De esta forma se pueden construir arrays multidimensionales.

Ejemplo 3.2.14. Se muestra como almacenar una matriz, o array de dos dimensiones, que contiene una sopa de letras generada

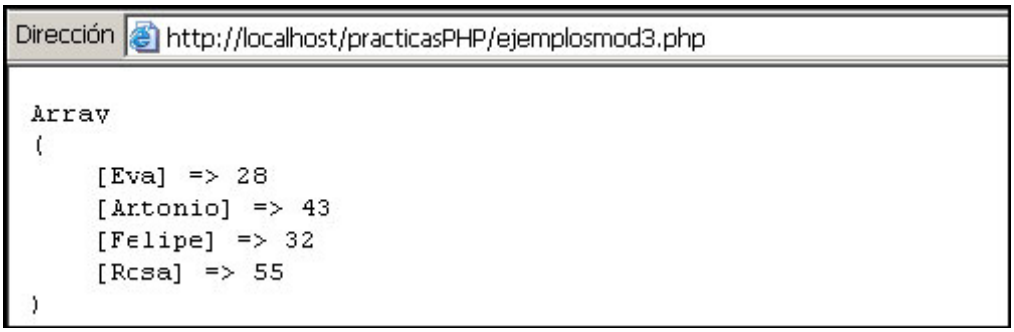
aleatoriamente, mediante la función *chr()*, que devuelve un carácter dado su número ascii.

```
<font face="Courier New">
<?php
  define("ALTO",10);
  define("ANCHO",20);
  $sopa_letras = array();
  for ($i = 0; $i < ALTO; $i++) {
    for ($j = 0; $j <ANCHO; $j++) {
      $sopa_letras[$i][$j] = chr(rand(65, 90));
    }
  }
  echo "SOPA DE LETRAS<br>";
  echo "<br>";
  for ($i = 0; $i < ALTO; $i++) {
    for ($j = 0; $j <ANCHO; $j++) {
      echo $sopa_letras[$i][$j];
    }
    echo "<br>";
  }
?>
</font>
```

3.2.6 Mostrar el contenido de un array.

Ejemplo 3.2.15. *PHP* dispone de la función *print_r()* a la que se le pasa como argumento un array, e imprime por pantalla su contenido. Para poderlo ver bien se usa la etiqueta HTML “<pre>... </pre>” que sirve para mostrar el texto y los saltos de línea tal cual se ha escrito.

```
<pre>
<?php
  $edades = array("Eva" => 28,
                  "Antonio" => 43,
                  "Felipe" => 32,
                  "Rosa" => 55);
  print_r($edades);
?>
</pre>
```



3.2.7 Desbordamiento de arrays.

Si se intenta acceder a un elemento que no existe en un array *PHP* nos devolverá la cadena vacía (""). Esta es una causa de error bastante

habitual, especialmente si después vamos a hacer operaciones con el dato extraído.

Ejemplo 3.2.16. Aquí se accede al quinto elemento del array que, en realidad, no existe.

```php
<?php
  $formas = array("triángulo", "cuadrado", "círculo");
  echo "El cuarto valor es: '" . $formas[4] . "'";
?>
```

3.2.8 Sentencia foreach.

A partir de la versión 4 de *PHP* se incluyó la sentencia de control *foreach* que permite recorrer arrays de forma cómoda. Hay dos sintaxis disponibles. La primera es un bucle que recorre todos los elementos del array. En cada ciclo la variable *$valor* toma el valor del elemento actual.

```php
foreach(array as $valor) {
  // bloque de código
}
```

La segunda sintaxis es similar, solo que además de tomar la variable *$valor* el valor del elemento actual, la variable *$key* tomará el valor del índice (*key* o clave).

```php
foreach(array as $key => $valor) {
  // bloque de código
}
```

Resulta especialmente útil cuando se usa en conjunción con arrays asociativos.

3.3 Funciones.

PHP es un lenguaje estructurado, y como tal dispone de funciones. Las funciones no son más que fragmentos de código, que podrían verse como pequeños programas, y que pueden ejecutarse posteriormente en resto del código. Gracias a esto permiten:

- Reutilizar código que se usa frecuentemente.
- Estructurar lógicamente el código de la aplicación para que sea más comprensible.
- Separar el código en diferentes ficheros para poder compartirlo en diferentes páginas *PHP*.

3.3.1 Funciones definidas por el usuario.

3.3.1.1 Declaración de una función.

Antes de poder utilizar una función ésta debe ser escrita. Por lo tanto, el lugar en el que se suelen situar las funciones es al comienzo del archivo, de tal forma que estén disponibles a partir de ese momento. Situarlas al comienzo también ayuda a una estructuración más lógica del código.

Las funciones se declaran, al igual que las variables, con un nombre. Éste irá precedido de la palabra *function*, unos paréntesis (para albergar los datos de entrada, llamados argumentos o parámetros) y unas llaves que incluyen el cuerpo de la función. Dentro del cuerpo de la función podemos declarar variables, llamadas a otras funciones y demás sentencias.

```
function nombre_funcion (arg1, arg2, arg3, ...) {
  // bloque de código
  return valor; //Opcional
}
```

3.3.1.2 Llamada a una función.

Para utilizar una función se escribe su nombre, seguido de paréntesis y dentro de estos se escriben los datos que se quieren pasar a la función (o variables que contienen dichos datos).

```
nombre_funcion (arg1, arg2, arg3, ...);
```

Si la función devuelve un dato, éste se puede asignar a una variable o usar directamente como parte de una expresión.

Ejemplo 3.3.1. A continuación se muestra una función muy sencilla, que no tiene argumentos.

```
<?php
  function hola_mundo() {
    echo "¡Hola Mundo!";
  }

  hola_mundo();
?>
```

3.3.1.3 Devolución de un valor.

Las funciones son mucho más útiles si pueden devolver un dato. Para ello usan la sentencia *return*. Una vez que se llega a esta instrucción no se ejecuta el código que se pueda encontrar a continuación.

Ejemplo 3.3.2. La siguiente función devuelve aleatoriamente un día de la semana.

```php
<?php
  function dia_semana() {
    $semana = array("lunes", "martes", "miércoles",
                    "jueves", "viernes", "sabado", "domingo");
    $dia = $semana[rand(0, 6)];
    return $dia;
  }

  $dia_cine = dia_semana();
  echo "El próximo $dia_cine voy al cine.";
?>
```

3.3.1.4 Argumentos.

Aún así, estas funciones que hemos visto son muy sencillas y no permiten hacer gran cosa. Lo más habitual es que a las funciones se les pasen datos, para que luego operen con éstos, y que al terminar la función devuelva el resultado. A los datos que recibe una función se les llama argumentos o parámetros.

En la declaración de la función, tras la palabra clave *function*, va el nombre de la función seguido por una lista de argumentos entre paréntesis y separados por comas. Dentro del cuerpo de la función estos datos se pueden utilizar como una variable cualquiera.

Ejemplo 3.3.3. Función sencilla que recibe un parámetro y devuelve un resultado. Luego es llamada dentro de un bucle, para mostrar el cuadrado de los números del 1 al 10.

```php
<?php
  function cuadrado($numero) {
    return $numero * $numero;
  }

  for ($i = 1; $i <= 10; $i++) {
    echo "$i al cuadrado es igual a " . cuadrado($i) . ".<br>";
  }
?>
```

3.3.1.5 Paso de argumentos por valor.

Cuando se pasan valores a las funciones, podemos pasarlos de 2 formas distintas: por valor y por referencia. El comportamiento predefinido en *PHP* se conoce como paso por valor.

En el paso de argumentos por valor, la variable que recibe el valor hace una copia del mismo, convirtiéndose en una nueva variable, y por tanto actuando a partir de ese momento como una variable totalmente independiente.

El aspecto práctico más importante consiste en que si cambiamos en algún momento el valor del argumento dentro de la función (copia), no se ve modificado fuera de ella (original).

3.3.1.6 Paso de argumentos por referencia.

En ocasiones es preferible no hacer la copia del dato que se pasa, sino trabajar directamente sobre el dato original.

Esto, por ejemplo, puede interesar cuando se trabaje con arrays, ya que el copiado de muchos datos puede perjudicar al tiempo de ejecución del programa, o más comúnmente cuando se quiere alterar el valor de la variable para obtener algún efecto.

A esto se le llama paso de parámetros por referencia. Para indicar que un parámetro se pasa por referencia se antepone el símbolo ampersand, "&", en la declaración de la función.

Una misma función puede emplear argumentos pasados por valor y por referencia, en una misma declaración.

Ejemplo 3.3.4. Ejecutando el siguiente código podemos observar los diferentes resultados obtenidos entre utilizar la misma función empleando un paso de parámetros por valor o por referencia. Como es de esperar, mediante el paso por referencia la variable exterior se verá afectada por los cambios ocurridos en el interior de la función.

```php
<?php
  function duplicar_por_valor($argumento) {
    $argumento = $argumento * 2;
    echo "Dentro de la función vale $argumento.<br>";
  }

  function duplicar_por_referencia(&$argumento) {
    $argumento = $argumento * 2;
    echo "Dentro de la función vale $argumento.<br>";
  }

  $numero1 = 5;
  echo "Antes de llamar a la función vale $numero1.<br>";
  duplicar_por_valor($numero1);
  echo "Después de llamar a la función vale $numero1.<br>";
  echo "<br>";

  $numero2 = 7;
  echo "Antes de llamar a la función vale $numero2.<br>";
  duplicar_por_referencia($numero2);
  echo "Después de llamar a la función vale $numero2.<br>";
?>
```

Ejemplo 3.3.5. Aunque el lenguaje no nos permite hacer que una función devuelva dos valores, tenemos la posibilidad de pasar por referencia las variables a devolver y, así, modificar su contenido en el cuerpo de la función.

Aquí se emplea esta estratagema para intercambiar el contenido de dos variables.

```php
<?php
  function intercambiar(&$argumento1, &$argumento2) {
    $auxiliar = $argumento1;
    $argumento1 = $argumento2;
    $argumento2 = $auxiliar;
  }

  $numero1 = 5;
  $numero2 = 8;
  echo "Antes: ($numero1, $numero2)<br>";
  intercambiar($numero1, $numero2);
  echo "Despues: ($numero1, $numero2)";
?>
```

3.3.1.7 Alcance de las variables.

Dentro de las funciones también podemos declarar nuevas variables, pero, ¿que pasa si hay una variable dentro de una función que se llama igual que una variable externa? La respuesta es que la variable interna es diferente de la que está fuera, y por lo tanto su valor será totalmente independiente.

A esto se le llama *alcance de una variable*, y tiene las siguientes implicaciones:

- Las variables que se declaran dentro de una función sólo existen en el cuerpo de dicha función.
- Las variables globales, que son aquellas que no se han declarado dentro de ninguna función, son accesibles en cualquier momento, independientemente del punto del código en que nos encontremos.
- Las variables locales, que son las declaradas dentro de una función, tienen preferencia sobre las variables globales.

Ejemplo 3.3.6. Este código muestra el alcance de una variable dentro de una función.

No hay que confundir este ejemplo con el ejemplo 3.3.4. En este caso no hay ningún argumento.

```php
<?php
  function mi_ciudad() {
    $ciudad = "Madrid";
    echo "Dentro de la función vale $ciudad.<br>";
  }

  $ciudad = "Barcelona";
  echo "Antes de llamar a la función vale $ciudad.<br>";
  mi_ciudad();
  echo "Después de llamar a la función vale $ciudad.<br>"
?>
```

Ejemplo 3.3.7. Puede darse el caso de que queramos acceder a una variable global dentro del cuerpo de la función. Para conseguirlo le antepondremos la palabra clave *global* a la primera referencia de la variable, con lo que el interprete *PHP* sabe que estamos llamando a la variable externa.

```php
<?php
  function mi_ciudad() {
    global $ciudad;
    $ciudad = "Madrid";
    echo "Dentro de la función vale $ciudad.<br>";
  }

  $ciudad = "Barcelona";
  echo "Antes de llamar a la función vale $ciudad.<br>";
  mi_ciudad();
  echo "Después de llamar a la función vale $ciudad.<br>"
?>
```

No obstante, es recomendable usar en las funciones nombres de variables diferentes a los de las variables del programa principal.

Tampoco es conveniente usar variables globales dentro de las funciones. Es mejor pasar estas variables como parámetro, ya que mejora notablemente la fiabilidad y claridad del código.

3.3.2 Creación de bibliotecas.

Para conseguir un código lo más claro posible, es deseable que éste sea breve. Una forma de conseguirlo es extraer las funciones que se declaran a un archivo independiente con extensión "*.php*". Una ventaja de esta estrategia es que se pueden hacer accesibles estas funciones a más de una página en *PHP*.

Ejemplo 3.3.8. A continuación se muestra una biblioteca de funciones, guardada en el archivo "*utils.php*", en la que se ha optado por comentarios, para mejorar la legibilidad y mantenibilidad del código.

```php
<?php
  // Biblioteca de funciones de usuario: utils.php
  // Devuelve el argumento elevado al cuadrado
  function cuadrado($numero) {
    return $numero * $numero;
  }

  // Devuelve la raíz cuadrada del argumento
  function raiz($numero) {
    return sqrt($numero);
  }

  // Devuelve verdadero si el número es igual o mayor que cero
  function es_positivo($numero) {
    return ($numero >= 0);
  }
?>
```

Ejemplo 3.3.9. En los ficheros de biblioteca se puede incluir cualquier tipo de código.

También es posible crear y encontrarse archivos de configuración o de constantes predefinidas.

En este caso se guarda en el archivo "*config.php*".

```php
<?php
  // Fichero de configuración
  // Archivo config.php
  define("PI", 3.1416);
  define("NUMERO_E", 2.7183);
  define("EULER", 0.5772);
?>
```

3.3.2.1 Las instrucciones include() y require().

Para tener disponibles las funciones de un fichero externo hay que indicarle al código *PHP* que debe incorporarlas al script actual.

Mediante estas instrucciones se incluye el contenido del fichero importado en el punto exacto en el que realiza la importación.

La diferencia entre las funciones de importación disponibles reside en que *require()* lanza un error fatal en el caso de no encontrar el fichero a importar, mientras que *include()* no lo hace.

Ejempo 3.3.10. La inclusión de bibliotecas se suele poner al principio del código para que estén disponibles en el resto del código.

```php
<?php
  include("config.php");
  include("utils.php");
  $radio = 5;
  $circunferencia = 2 * $radio * PI;
  $area = PI * cuadrado($radio);
  echo "Un círculo de rádio $radio tiene circunferencia ";
  echo "$circunferencia y área $area.<br>";
  $area = -8;
  if (es_positivo($area)) {
    $radio = raiz($area / PI);
    echo "Un círculo de área $area tiene un rádio $radio";
  } else {
    echo "No se puede calcular, área negativa.";
  }
?>
```

Hay que tener en cuenta el orden en el que se cargan los archivos externos, especialmente si unos hacen uso del código de otros.

3.3.3 Otros tipos de funciones.

Hasta ahora hemos visto las funciones que nosotros definimos. Pero a lo largo de muchos ejemplos hemos ido usando funciones como *rand()* o

number_format() que ya estaban disponibles en el lenguaje. Éstas son funciones del núcleo de *PHP*, y por lo tanto no hace falta hacer un *include()* para poder usarlas.

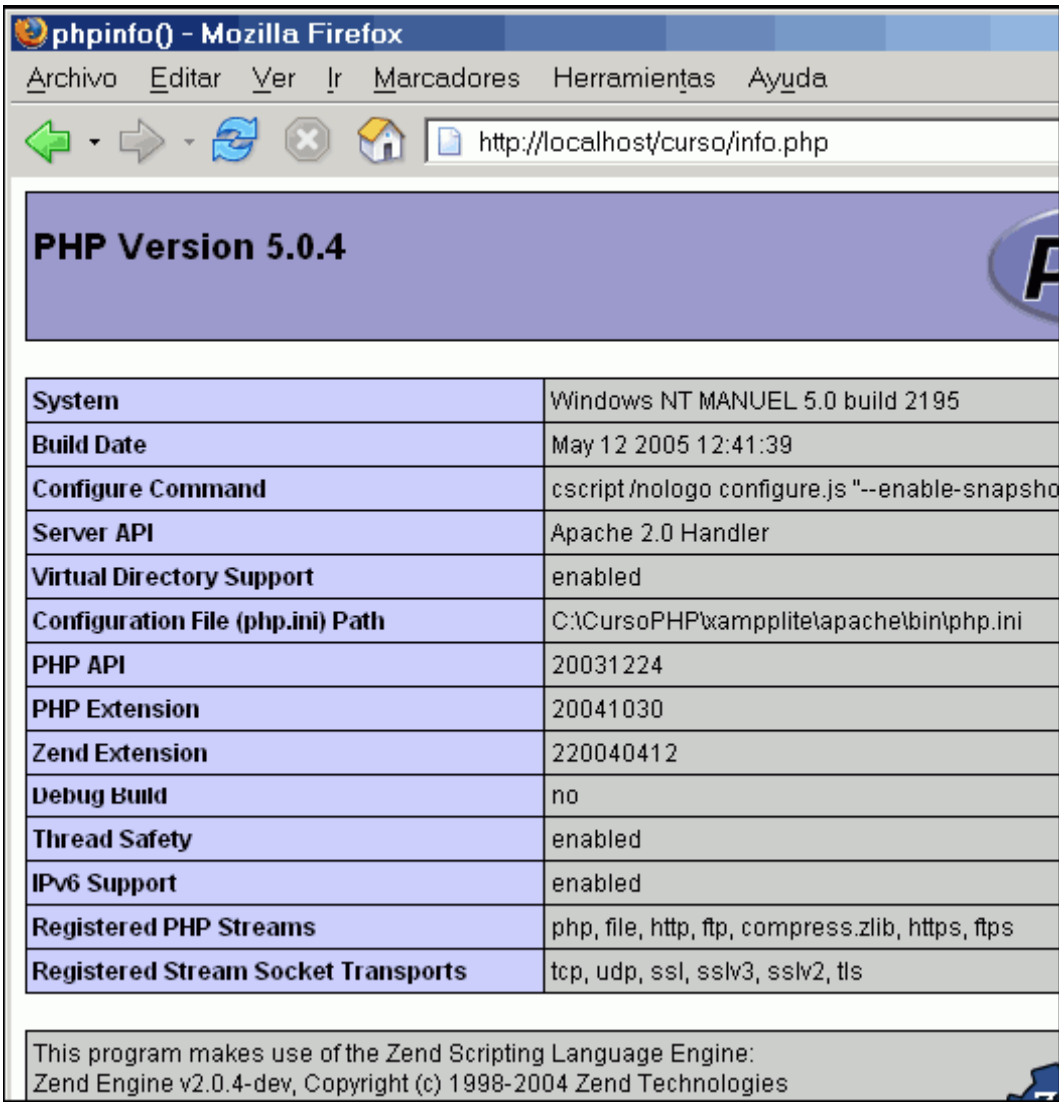
3.3.3.1 Extensiones de PHP.

Las funciones que incorpora de forma predefinida *PHP* están agrupadas en extensiones.

Ejemplo 3.3.11. Para saber que extensiones están instaladas en nuestro interprete de *PHP* tenemos una opción muy útil, la función *phpinfo()* que genera una página con la información del intérprete que tenemos instalado.

```
<?php
  phpinfo();
?>
```

Debe aparecernos una pantalla similar a la siguiente:



3.3.3.2 Referencia sobre funciones.

Toda la referencia sobre funciones se puede encontrar en la página oficial sobre *PHP*: http://es.php.net/manual/es/funcref.php.

Hay una gran multitud de funciones en *PHP* de entre las cuales probablemente usaremos sólo unas pocas. Las extensiones más útiles son:

- Funciones matemáticas: http://es.php.net/manual/es/ref.math.php.
- Funciones de cadenas: http://es.php.net/manual/es/ref.strings.php.
- Funciones de mysql: http://es.php.net/manual/es/ref.mysql.php.
- Funciones de fecha y hora: http://es.php.net/manual/es/ref.datetime.php.

La página dispone también de un buscador que resulta bastante útil.

3.4 Formularios.

Hasta este momento todas las páginas que hemos visto a través de ejemplos tenían un defecto, no eran interactivas. Únicamente hacían operaciones en función de los datos de los que ya disponían, pero eran incapaces de recibir datos del navegador. Es decir, de un usuario externo al propio sistema.

Una de las formas más habituales de recibir datos por parte de una página web dinámica es a través de los formularios web. Estrictamente hablando, los formularios no son parte del lenguaje *PHP* sino del lenguaje *HTML*.

Sin embargo, y aunque hay algunas excepciones, los formularios *HTML* suelen ser recibidos por páginas dinámicas encargadas de procesarlos y/o almacenarlos.

Hay que tener en cuenta que además de formularios *HTML* hay otras tecnologías que permiten el envío de datos a páginas web, con Flash u otras páginas web dinámicas.

3.4.1 Formularios HTML.

Los formularios pueden llegar a ser muy complejos, con una gran cantidad de campos de entrada, validación de datos por *Javascript*,

campos ocultos, y otros. Pero el objetivo del curso no es explicar su funcionamiento a fondo, así que se hará un repaso muy superficial.

Un formulario está formado por tres tipos de elementos: etiquetas de comienzo y final, campos de entrada de datos y botones para realizar acciones.

3.4.1.1 Declaración del formulario.

Un formulario se distingue por estar encuadrado entre una etiqueta de comienzo de formulario "*<form ...>*" y una de final de formulario "*</form>*". La etiqueta de comienzo de formulario ha de tener al menos tres atributos.

- *name*: Aunque no es obligatorio, es muy recomendable darle un nombre al formulario para que sea fácil de identificar.
- *method*: Este parámetro puede tomar dos valores según la forma de enviar los datos:
    - *get*: los datos se envían codificados en la URL o dirección de la página destino.
    - *post*: los datos se envían ocultos dentro del contenido de la petición.
- *action*: indica la página destino que se encargará de procesar los datos que envía el formulario actual.

```
<form name="formulario" method="get" action="procesar.php">
  Campos de entrada de datos
  Botones de acciones
</form>
```

3.4.1.2 Campos de los formularios.

Una vez declarado el formulario hemos de poner un campo para cada uno de los datos que queremos recibir. Hay 6 tipos de campos:

- Texto: se usa para recibir una línea de texto.

```
<input type="text" name="nombre">
```

- Selección única: se usa para elegir una opción entre varias. Si se quiere que las opciones sean excluyentes entre sí han de tener el mismo nombre, y diferentes valores.

```
<input type="radio" name="sexo" value="hombre">
<input type="radio" name="sexo" value="mujer">
```

- Selección múltiple: se usa para elegir una o más opciones de un grupo.

```
<input type="checkbox" name="cine">
<input type="checkbox" name="musica">
<input type="checkbox" name="lectura">
```

- Lista de selección: es un sistema alternativo al campo de selección única para elegir una opción entre varias, esta vez con formato de lista desplegable.

```
<select name="sexo">
  <option>hombre</option>
  <option>mujer</option>
</select>
```

- Área de texto: sirve para poder recibir textos largos compuestos por varias líneas.

```
<textarea name="opinion"></textarea>
```

- Campo oculto: en ocasiones queremos pasar un dato a la página *PHP* que procesará el formulario, pero no queremos que esté se muestre en un campo visible. Puede ser útil para pasar información entre páginas *PHP* independientes.

```
<input type="hidden" name="referencia">
```

En todos ellos es importante utilizar el atributo *name*, que contendrá el nombre de variable con el cual vamos a recibir los datos. El atributo *value* es opcional e indica el valor predefinido que se les da.

3.4.1.3 Botones de los formularios.

Todos los formularios han de tener un botón para enviar los datos una vez que se han rellenado.

```
<input type="submit" name="Enviar">
```

También se suele incluir un botón para devolver los campos del formulario a su estado y/o valor inicial (puede que vacíe el campo).

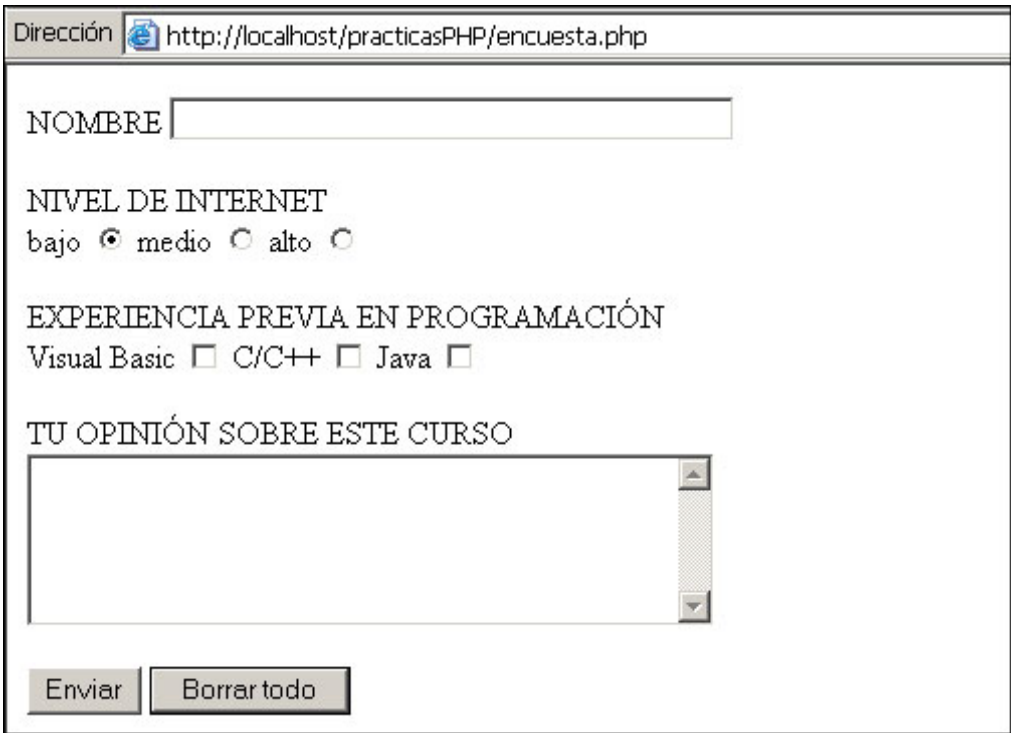
```
<input type="reset" name="Borrar todo">
```

También disponemos de los botones de tipo "*button*" que se emplean para introducir código de la parte cliente, generalmente *Javascript*. Se suelen usar para facilitar y/o verificar parte o la totalidad de la información contenida en el formulario. Sin embargo, no es objetivo de este curso ahondar en el lenguaje *Javascript*, y sus diversos usos.

Ejemplo 3.4.1. A continuación se muestra un formulario formado por algunos de los tipos de campo descritos con anterioridad y se guarda en un archivo llamado "*encuesta.php*".

```
<html>
  <head>
    <title></title>
  </head>
  <body>
    <form name="encuesta" method="get" action="verificar.php">
      NOMBRE <input type="text" name="nombre" size="43"><br>
      <br>
      NIVEL DE INTERNET<br>
      bajo <input type="radio" name="nivel" value="bajo" checked>
      medio <input type="radio" name="nivel" value="medio">
      alto <input type="radio" name="nivel" value="alto"><br>
      <br>
      EXPERIENCIA PREVIA EN PROGRAMACIÓN<br>
      Visual Basic <input type="checkbox" name="basic">
      C/C++ <input type="checkbox" name="c_cplus">
      Java <input type="checkbox" name="java"><br>
      <br>
      TU OPINIÓN SOBRE ESTE CURSO<br>
      <textarea name="opinion" cols="40" rows="5"></textarea><br>
      <br>
      <input type="submit" value="Enviar">
      <input type="reset" value="Borrar todo">
    </form>
  </body>
</html>
```

Deberíamos ver un formulario similar a este:



3.4.2 Recepción de datos de formularios.

Recibir los datos no puede ser más sencillo. El script de destino tendrá definida un variable por cada campo con el mismo nombre que se le haya dado en el formulario. Y esta variable contendrá el valor que se haya introducido en el campo. Esta variable ya está disponible desde el comienzo del script, no hay que hacer nada especial, sólo leer su contenido.

Ejemplo 3.4.2. Esta página *PHP* recibe los datos de la encuesta y los muestra. Para que funcione hay que guardarla con el nombre "*verificar.php*", que es el que aparecía en el atributo *action* del formulario.

```php
<?php
  echo "Comprueba si tus datos son correctos.<br>";
  echo "<br>";
  echo "{$_REQUEST["nombre"]}.<br>";
  echo "Nivel de internet: {$_REQUEST["nivel"]}.<br>";
  if (($_REQUEST["basic"] == "on")
       || ($_REQUEST["c_cplus"] == "on")
       || ($_REQUEST["java"] == "on")) {
    echo "Con experiencia en ";
    if ($_REQUEST["basic"] == "on") {
      echo "Visual Basic";
      if ($_REQUEST["c_cplus"] == "on") {
        echo ", C/C++";
      }
      if ($_REQUEST["java"] == "on") {
        echo ", Java";
      }
    } else if ($_REQUEST["c_cplus"] == "on") {
      echo "C/C++";
      if ($_REQUEST["java"] == "on") {
        echo ", Java";
      }
    } else {
      if ($_REQUEST["java"] == "on") {
        echo "Java";
      }
    }
    echo ".<br>";
  } else {
    echo "Sin experiencia previa en programación.<br>";
  }
  echo "<br>";
  echo "OPINIÓN SOBRE EL CURSO:<br>";
  echo nl2br($_REQUEST["opinion"]); //convierte saltos de línea en <br>
?>
```

Ejemplo 3.4.3. Los parámetros que nos han llegado se encuentran almacenados en un array asociativo con nombre de variable $_REQUEST. Podemos ver su contenido usando la función *print_r()* que se vió en la unidad didáctica anterior. Aunque en algún caso se puede

acceder a estos datos sin $_REQUEST, resulta totalmente desaconsejado ya que su efectividad dependerá del navegador[1].

```
<pre>
<?php
  print_r($_REQUEST);
?>
</pre>
```

3.4.2.1 Páginas que envían, reciben y procesan datos.

En este apartado se muestra una práctica común mediante la que la página web que contiene el formulario, y la página web que recibe y procesa la información se fusionan en un mismo fichero, haciendo uso por tanto de una única página web.

Para aplicar esta técnica debemos tener en cuenta los siguientes aspectos:

- Debemos comprobar si recibimos alguno de los datos de formulario:
    - En caso de recibir alguno de los datos, los procesaremos.
    - Si no recibieramos los datos, mostraremos el formulario.
- Debemos enviar el formulario, mediante su atributo *action*, a la propia pagina web que contiene el formulario. Para ello, podemos emplear la variable predefinida *$PHP_SELF*, que contiene el nombre de archivo de la página web actual.

Sin embargo, esta práctica tan extendida tiene unas ventajas y unos inconvenientes que pueden no resultar siempre beneficiosos. Por un lado tenemos la ventaja de agrupar conceptualmente toda la acción en un único fichero, ya que con una única página web podremos contener todos los elementos necesarios para llevar a cabo una tarea. Sin embargo, este procedimiento dificulta considerablemente la separación de la programación y el diseño.

Por este motivo, resulta desaconsejable llevar a cabo esta práctica en aplicaciones que vayan a tener una envergadura considerable.

Ejemplo 3.4.4. Esta página web une el formulario de entrada de datos y la recepción de los mismos. Podemos guardar este archivo con el nombre "*juegoppt.php*".

[1] Para poder ver algún resultado, necesitaremos cambiar el parámetro *action* del formulario, o sustituir el fichero "*verificar.php*" por el código de este ejemplo.

```
<?php
  $opciones = array("piedra", "papel", "tijeras");
  if ($_REQUEST["jugada"] != "") {
    $mijugada = $opciones[rand(0, 2)];
    if ($_REQUEST["jugada"] == $mijugada) {
      $resultado = "Empates.";
    } else if (($_REQUEST["jugada"] == "piedra" && $mijugada == "tijeras")
           || ($_REQUEST["jugada"] == "tijeras" && $mijugada == "papel")
           || ($_REQUEST["jugada"] == "papel" && $mijugada == "piedra")) {
      $resultado = "Tu ganas.";
    } else {
      $resultado = "Gano yo.";
    }
  }
?>

<html>
  <head>
    <title>Piedra, papel o tijera</title>
  </head>
  <body>
<?php
  if ($_REQUEST["jugada"] != "") {
?>
    Has elegido <?= $_REQUEST["jugada"]; ?>, yo he elegido <?= $mijugada ?>.
    <?= $resultado ?><br>
    ¿Quieres jugar otra vez?<br>
<?php
  }
?>
    <form "juego" method="post" action="<?= $PHP_SELF ?>">
      Piedra<input type="radio" name="jugada" value="piedra">
      Papel<input type="radio" name="jugada" value="papel">
      Tijera<input type="radio" name="jugada" value="tijeras">
      <input type="submit" value="Jugar">
    </form>
  </body>
</head>
```

El código que se incluye al principio sólo se ejecuta si el dato *$jugada* no llega vacío. Es decir, cuando se ha pulsado el botón de enviar.

3.4.2.2 Redirección de páginas web.

En ocasiones se necesita que un programa *PHP* reciba datos y haga una operación "silenciosa" con los mismos. Es decir, que no sea necesario que se muestre una página al navegador sino que se vuelva a la inicial, como un formulario que envía, recibe y procesa datos.

Una alternativa es hacer una página que procese los datos y que al terminar se redirija automáticamente a otra. Para poder hacerlo *PHP* nos ofrece la función *header()*, que sirve para enviar cabeceras *HTML*, y que en el caso concreto que nos ocupa, se usa de la siguiente forma:

```
header("location:url")
```

La dirección de destino *url* puede ser una ruta absoluta o relativa.

Es imprescindible que la página no contenga al principio código *HTML*, ya que las cabeceras deben enviarse siempre al principio del documento. De hecho, aquí nos encontramos un error muy habitual, ya que tan sólo con que se nos haya descuidado un espacio o salto de linea a enviar como *HTML*, ocasionará un error.

También hay que tener en cuenta que una vez que se llega a la función *header* no se sigue ejecutando el resto de la página, por lo que las operaciones que queramos hacer las deberemos ejecutar antes.

Ejemplo 3.4.5. Este es un ejemplo típico de recepción de datos. En primer lugar, se incluye una sencilla página web con un formulario que pide el nombre y un comentario para almacenarlo en un registro de visitas. En el script "*insertar.php*" se almacena en la base de datos y se redirecciona a una página para dar las gracias.

- visita.html

```
<html>
  <head>
    <title>Libro de visitas</title>
  </head>
  <body>
    <form name="visita" method="post" action="insertar.php">
      Nombre: <input type="text" name="nombre" size="44"><br>
      Comentario:<br>
      <textarea name="comentario" cols="40" rows="5"></textarea><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

- insertar.php

```
<?php
  // Aquí vendría el código para guardar en la base de datos
  header("location:gracias.html"); //Redirección
?>
```

- gracias.html

```
<html>
  <head>
    <title>Gracias</title>
  </head>
  <body>
    Gracias por su participación.
  </body>
</html>
```

Ejemplo 3.4.6. Uno de sus usos habituales es redirigir hacia una página de error, en caso de que haya fallos en los datos. Aquí se valida una dirección de email, y si es incorrecta se reenvía a una página de error.

- suscripcion.html

```
<html>
  <head>
    <title>Suscripción al boletín electrónico</title>
  </head>
  <body>
    Bienvenido al servicio de suscripción.<br>
    Por favor, introduzca una dirección de email válida.<br>
    <form name="suscripcion" method="post" action="guardar.php">
      Email: <input type="text" name="email" size="40"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

- guardar.php

```
<?php
  // Indica la posición del caracter "@" o FALSE si no está
  $posicion_arroba = strpos($_REQUEST["email"], "@");
  // Busca la aparición de un punto (.) a partir de la arroba
  $posicion_punto = strpos($_REQUEST["email"], ".", $posicion_arroba);
  if ($posicion_arroba && $posicion_punto) {
    // Aquí vendría el código para guardar en la base de datos
    header("location:confirmacion.html"); // Redirección
  } else {
    // Aquí vendría el código para guardar en la base de datos
    header("location:error.html");        // Redirección
  }
?>
```

- confirmacion.html

```
<html>
  <head>
    <title>Confirmacion</title>
  </head>
  <body>
    Su suscripción ha sido registrada.</br>
    Gracias por su interés.
  </body>
</html>
```

- error.html

```
<html>
  <head>
    <title>Error</title>
  </head>
  <body>
    Introdujo una dirección de correo no válida.<br>
    Por favor vuelva a solicitar la suscripción.
    <a href="suscripcion.html"> Volver </a>
  </body>
</html>
```

3.5 Manejo de ficheros.

Mediante los formularios hemos conseguido aportar interactividad a nuestras páginas web. Pero otra de las limitaciones fundamentales de lo que podemos hacer con lo que sabemos hasta el momento es que una vez que se cierra una página web los datos que contiene se pierden.

En el próximo módulo veremos como guardar la información en una base de datos.

Esto es útil para grandes cantidades de información muy estructurada, pero hay casos en los que la información que queremos guardar es pequeña. En este apartado veremos como podemos usar *PHP* para acceder al sistema de archivos del servidor para poder leer o guardar datos.

3.5.1 Manejo de ficheros.

3.5.1.1 Abrir.

Para acceder a un archivo primero es necesario abrirlo. Para ello usaremos la función *fopen() que* tiene dos argumentos, el nombre del archivo a acceder y el modo de acceder a este.

```
fopen(ruta al archivo, modo de acceso)
```

PHP ofrece los siguientes modos de acceso:

Modo	Descripción
r	Apertura para sólo lectura; ubica el apuntador de archivo al comienzo del mismo.
r+	Apertura para lectura y escritura; ubica el apuntador de archivo al comienzo del mismo.
a	Apertura para sólo escritura; ubica el apuntador de archivo al final del mismo. Si el archivo no existe, intenta crearlo.
A+	Apertura para lectura y escritura; ubica el apuntador de archivo al final del mismo. Si el archivo no existe, intenta crearlo.
w	Apertura para sólo escritura. Cualquier contenido del archivo será borrado. Si el archivo no existe, intenta crearlo.
W+	Apertura para lectura y escritura. Cualquier contenido del archivo será borrado. Si el archivo no existe, intenta crearlo.

Se llama apuntador a la posición del archivo en la que leemos o escribimos. Como podemos ver lo más habitual es que lo situemos al principio para leer todo su contenido o al final para ir añadiendo datos.

La función devuelve un manejador que usaremos posteriormente para manipular el archivo (y que guardaremos en una variable), o devuelve FALSE si no ha podido acceder por alguna causa (permisos, ruta, memoria, etc).

3.5.1.2 Cerrar.

Mientras hacemos operaciones con el archivo lo debemos mantener abierto, pero al terminar de trabajar con él hay que cerrarlo para que el sistema operativo pueda disponer de él, ya que mientras está abierto el sistema operativo lo bloquea para que otros programas no puedan escribir y destruir mutuamente lo que escriben.

Para cerrar un archivo abierto se usa la función *fclose()* pasándole como parámetro la variable que contiene el manejador del archivo.

```
fclose(archivo)
```

Ejemplo 3.5.1. En este código se abre un fichero en modo de lectura. En función de si se ha conseguido, se muestra un mensaje de confirmación o de error. Finalmente el archivo se cierra.

```
<?php
  $ruta = "utils.php";
  $archivo = fopen($ruta, "r");
  if ($archivo) {
    print "Archivo $ruta abierto para lectura.";
  } else {
    print "No se pudo abrir el archivo: $ruta.";
  }
  fclose($archivo);
?>
```

Ejemplo 3.5.2. El fichero que abrimos lo podemos localizar mediante una ruta absoluta o relativa. Aunque en *Windows* las rutas se construyan usando la contrabarra " \ ", *PHP* admite que se separen los directorios con la barra normal " / " como en *Linux*. Esta última es la forma que elegiremos ya que las rutas relativas que construyamos de esta forma serán válidas tanto si instalamos nuestra aplicación en *Linux* como en *Windows*.

```
<?php
  $ruta_absoluta = "c:/CursoPHP/htdocs/index.php";
  $ruta_relativa = "../practicasPHP/config.php";
  $archivo1 = fopen($ruta_absoluta, "r");
  $archivo2 = fopen($ruta_relativa, "r");
  fclose($archivo1);
  fclose($archivo2);
?>
```

Ejemplo 3.5.3. Es incluso posible abrir archivos que estén alojados en otros servidores, aunque lo más habitual es que solo tengamos permisos de lectura.

```php
<?php
  $url = "http://www.google.es/index.html";
  $archivo = fopen($url, "r");
  fclose($archivo);
?>
```

3.5.1.3 Leer.

Lo más habitual es que queramos leer un archivo. *PHP* ofrece muchas formas de hacerlo. Una de las más sencillas es mediante la función *fread()* que lee un número de caracteres de un archivo. En conjunción con la función *filesize()* que nos devuelve el tamaño del archivo en bytes se puede usar para leer todo el archivo.

```
fread(archivo, tamaño)
```

Ejemplo 3.5.4. Lectura del archivo "*prueba.txt*". Para que el archivo funcione tendremos que haberlo creado en la misma carpeta que este script, con el contenido que deseemos.

```php
<pre>
<?php
  $archivo = fopen("prueba.txt", "r");
  $tamano = filesize("prueba.txt");
  $texto = fread($archivo, $tamano);
  echo $texto;
  fclose($archivo);
?>
</pre>
```

3.5.1.4 Escribir.

Otra acción que querremos hacer habitualmente es añadir datos a un archivo. Para ello se usa una función muy sencilla *fwrite()*, que escribe en la posición en la que está el apuntador. Por lo general, si hemos abierto el archivo en modo "a" escribiremos al final del archivo.

La función *fwrite()* usa dos parámetros. El primero el manejador del archivo y el segundo la cadena que queremos escribir.

```
fwrite(archivo, cadena de texto)
```

Ejemplo 3.5.5. En este ejemplo se usa esta función dos veces para escribir dos frases en el archivo (Veremos que el fichero se crea al acceder a la página del script con el navegador).

```php
<?php
  $archivo = fopen("refranes.txt", "a");
  fwrite($archivo, "Si las barbas de tu vecino ves cortar ...\r\n");
  fwrite($archivo, "...pon las tuyas a remojar \r\n");
  fclose($archivo);
?>
```

La cadena " \r\n " sirve para insertar un salto de línea en el archivo en el sistema operativo *Windows*. En *Linux* se usa solo la cadena " \n ".

Si lo que queremos es sobrescribirlo abriremos el archivo en modo "w", de tal forma que se borre el contenido al abrirlo y dispondremos de un fichero vacío.

3.5.1.5 Crear un fichero.

Si queremos crear un archivo bastará que lo abramos con el modo "a" o "w". Al abrirlo, si el archivo no existe lo creará.

3.5.1.6 Eliminar un fichero.

La función *unlink()* que recibe como parámetro una ruta a un fichero lo borra. En el caso de que no lo haya conseguido, por no tener permisos o sencillamente porque no existe el archivo, devuelve FALSE.

```
unlink(archivo)
```

Ejemplo 3.5.6. Este programa intenta borrar un archivo y en el caso de no conseguirlo muestra un mensaje de error.

```
<?php
  if (!unlink("refranes.txt")) {
    echo "No se ha podido borrar el archivo.";
  }
?>
```

3.5.2 Atributos de un fichero.

Para manejar de forma segura es conveniente que conozcamos sus atributos. Por ejemplo, antes de abrir un archivo para escritura convendría comprobar si tenemos permisos de lectura. Las siguientes funciones nos ayudarán en esta tarea.

3.5.2.1 file_exists().

Devuelve TRUE si el archivo por el que preguntamos existe.

3.5.2.2 file_size().

Ya ha aparecido al hablar de la lectura de un archivo. Devuelve el tamaño en bytes del mismo.

3.5.2.3 is_file().

Devuelve TRUE si el archivo es un fichero.

3.5.2.4 is_dir().

Devuelve TRUE si el archivo es un directorio.

3.5.2.5 is_readable().

Devuelve TRUE si el archivo se puede abrir para lectura.

3.5.2.6 is_writeable().

Devuelve TRUE si el archivo se puede abrir para escritura.

Ejemplo 3.5.7. Mediante este programa se muestran las propiedades del archivo "*prueba.txt*".

```
<?php
  $ruta = "prueba.txt";
  if (file_exists($ruta)) {
    echo "$ruta tiene un tamaño de ";
    echo filesize($ruta) . " bytes.<br>";
    if (is_file($ruta)) {
      echo "$ruta es un fichero.<br>";
    }
    if (is_dir($ruta)) {
      echo "$ruta es un directorio.<br>";
    }
    if (is_readable($ruta)) {
      echo "$ruta se puede abrir para lectura.<br>";
    }
    if (is_writable($ruta)) {
      echo "$ruta se puede abrir para escritura.<br>";
    }
  } else {
    echo "$ruta no existe.";
  }
?>
```

3.5.3 Manejo de directorios.

PHP ofrece muchas funciones para manejar directorios, pero lo más probable es que en la mayoría de los casos lo único que nos interese de un directorio es conocer los archivos que tiene. Una vez que conozcamos este dato podremos construir rutas relativas a sus subdirectorios y a su vez listarlos y así de forma sucesiva.

3.5.3.1 Abrir un directorio.

De forma similar a como sucedía con los archivos, antes de trabajar con un directorio lo tendremos que abrir. Nos serviremos de la función *opendir()* que devuelve un manejador del directorio.

```
opendir(ruta al directorio)
```

3.5.3.2 Listar los archivos de un directorio.

Para conocer los ficheros y subdirectorios que contiene un directorio se usa la función *readdir().* Por cada vez que llamemos a esta función nos

devolverá el nombre de cada archivo encontrado o FALSE, si no quedan más archivos en el directorio.

```
readdir(directorio)
```

Ejemplo 3.5.8. Aquí se puede ver como listar todos los contenidos del directorio actual (referenciado mediante la ruta relativa ".") indicando a su vez si son ficheros o directorios.

```
<?php
  $directorio = opendir(".");
  while ($archivo = readdir($directorio)) {
    if (is_file($archivo)) {
      echo "$archivo es un fichero.<br>";
    }
    if (is_dir($archivo)) {
      echo "$archivo es un directorio.<br>";
    }
  }
?>
```

3.5.4 Subir archivos al servidor.

El envío de archivos desde el cliente es una de las posibilidades más atractivas para una aplicación web. Se pueden subir de esta forma fotos, un curriculum, un documento PDF, etc.

Para poder enviar documentos en un formulario necesitamos dos elementos:

- Incluir el atributo *enctype* con el valor *multipart/form-data* en el formulario de envío.
- Incluir un campo *input* del tipo *file* al que el cliente asociará el fichero que enviaremos en el formulario.

Ejemplo 3.5.9. Se va a explicar como subir un archivo a través de un ejemplo. En primer lugar necesitamos un formulario especial, que llamaremos "*subir.html*".

- subir.html

```
<html>
  <head>
    <title>Elija el archivo a subir</title>
  </head>
  <body>
    <form name="subir" method="post" action="guardar_fichero.php"
                                    enctype="multipart/form-data">
      Archivo a subir <input type="file" name="miarchivo"><br>
      <input type="submit" value="Subir el archivo">
    </form>
  </body>
</html>
```

Además de este formulario necesitaremos una página dinámica en *PHP* que reciba el archivo y lo guarde en el disco duro. Esto se hace en 3 pasos:

- El servidor guarda el archivo en un fichero temporal.
- Movemos el archivo en la posición definitiva usando la función *move_uploaded_file()*. Esto es imprescindible ya que pasado cierto tiempo el servidor web eliminará automáticamente el fichero temporal.
- Si la operación de mover el archivo falla mostramos un mensaje de error.

El código que se ha escrito en este caso, en el script *guardar.php* es:

- guardar_fichero.php

```php
<?php
  $temporal = $_FILES["miarchivo"]["tmp_name"];
  $destino = "uploads/" . $_FILES["miarchivo"]["name"];
  if (move_uploaded_file($temporal, $destino)) {
    echo "Archivo subido con éxito";
  } else {
    echo "Ocurrió un error, no se ha podido subir el archivo";
  }
?>
```

Hay que tener cuidado, ya que en este script se asume que el directorio *uploads* ya existe. Si no es así, la operación fallaría.

Fíjese que para acceder al fichero, *PHP* nos crea un vector llamado *$_FILES*, que contendrá todos los ficheros que ha enviado el usuario. De este modo, mediante el nombre del campo del fichero, que puede consultarse en el formulario correspondiente, podría acceder a cada uno de los ficheros enviados para realizar las acciones oportunas.