

Tabla de contenido

Tabla de contenido.....	1
Cadenas.....	5
Combinar código HTML con código PHP.....	5
Variables dentro de cadenas.....	5
Uso de la sentencia echo.....	2
Uso de la sintaxis <?= ... ?>.....	2
Fragmentos de HTML dentro de estructuras condicionales.....	3
Distribución del código.....	4
Manipulación de cadenas.....	5
Acceder a los caracteres de una cadena.....	6
strlen().....	7
strpos().....	7
substr().....	7
trim().....	8
str_replace().....	8
strtoupper() y strtolower().....	9
number_format().....	9
Arrays.....	13
Arrays indexados numéricamente.....	13
Inicializar un array.....	13
Acceder a un elemento de un array.....	14
Recorrer un array.....	15
Ordenar un array.....	16
Arrays no secuenciales.....	17
Arrays asociativos.....	17
Arrays no homogéneos.....	18
Arrays multidimensionales.....	18
Mostrar el contenido de un array.....	19
Desbordamiento de arrays.....	19
Sentencia foreach.....	21
Comprobación y eliminación de posiciones de Arrays.....	21
isset().....	21
unset().....	23
Funciones.....	24
Funciones definidas por el usuario.....	24

Declaración de una función.....	24
Llamada a una función.....	24
Devolución de un valor.....	26
Argumentos.....	26
Paso de argumentos por valor.....	26
Paso de argumentos por referencia.....	27
Alcance de las variables.....	28
Creación de bibliotecas.....	30
Las instrucciones include() y require().....	31
require_once e include_once.....	32
Otros tipos de funciones.....	33
Extensiones de PHP.....	33
Variables y funciones. Qué trae de nuevo PHP 7.....	34
Tipado fuerte en PHP: Descriptivo y Estricto.....	34
Tipado fuerte descriptivo.....	34
Tipado fuerte Estricto.....	35
Formularios.....	36
Formularios HTML.....	36
Declaración del formulario.....	36
Campos de los formularios.....	37
Botones de los formularios.....	38
Recepción de datos de formularios.....	40
Páginas que envían, reciben y procesan datos.....	41
Redirección de páginas web.....	42
Operador de fusión de null o Null coalesce en PHP.....	45
Explicación y ejemplos de null coalesce en PHP.....	45
Más de dos operandos.....	45
3, 4, 5 o más operandos.....	46
Ejemplo final.....	46
Un poco de Seguridad en el Envío de Datos. Sanitización de datos.Limpiar y validar variables GET y POST en PHP.....	47
¿Cuál es la solución para estos problemas?.....	47
Validación y desinfección de datos con filtros.....	47
Lista de filtros de sanitización.....	48
Vamos a lo práctico: filter_var_array.....	48
Manejo de ficheros.....	51
Manejo de ficheros.....	51
Abrir.....	51
Cerrar.....	52
Leer.....	53
Escribir.....	56

Crear un fichero.....	57
Eliminar un fichero.....	57
Copiar y Renombar un fichero.....	57
Atributos de un fichero.....	57
file_exists(ruta/nombre).....	58
file_size(ruta/nombre).....	58
is_file(ruta/nombre).....	58
is_dir(ruta/nombre).....	58
is_readable(ruta/nombre).....	58
is_writeable(ruta/nombre).....	58
is_writeable(ruta/nombre).....	58
filemtime(ruta).....	58
Manejo de directorios.....	59
Subir archivos al servidor.....	60
Array Superglobal \$_FILES.....	62
Subida de múltiples ficheros.....	62
Más ejemplos.....	63



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.

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*.

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

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

```

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*.

```

<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>
</html>

```

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>
```

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
    $autorizado = FALSE;
    if (!$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
    }
?>
```

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.

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 la función *echo* y *print()* pero hay unas cuantas más.

Cadenas en php

Podemos considerar que en php es un string cualquier cadena de caracteres entrecomilladas.

```
$cadenaDoble =" Hola que tal ";
$cadenaSimple = 'Muy bien y tu ';
```

Si bien en principio no existe ninguna diferencia entre comilla entre usar cadenas simples y dobles en realidad hay pequeños matices de cierta importancia.

1. Cadenas simples: Se consideran más eficientes.
2. Cadenas dobles: Si un [string](#) está delimitado con comillas dobles ("), PHP interpretará las siguientes secuencias de escape como caracteres especiales:

Secuencia	Significado
\n	avance de línea (LF o 0x0A (10) en ASCII)
\r	retorno de carro (CR o 0x0D (13) en ASCII)
\t	tabulador horizontal (HT o 0x09 (9) en ASCII)
\v	tabulador vertical (VT o 0x0B (11) en ASCII) (desde PHP 5.2.5)
\e	escape (ESC o 0x1B (27) en ASCII) (desde PHP 5.4.4)
\f	avance de página (FF o 0x0C (12) en ASCII) (desde PHP 5.2.5)
\\	barra invertida
\\$	signo de dólar
\"	comillas dobles
\[0-7]{1,3}	la secuencia de caracteres que coincida con la expresión regular es un carácter en notación octal, que silenciosamente desborda para encajar en un byte (p.ej. "\400" === "\000")
\x[0-9A-Fa-f]{1,2}	la secuencia de caracteres que coincida con la expresión regular es un carácter en notación hexadecimal
\u{[0-9A-Fa-f]+}	la secuencia de caracteres que coincida con la expresión regular es un punto de código de Unicode, la cual será imprimida al string como dicha representación UTF-8 del punto de código (añadido en PHP 7.0.0)

Acceder a los caracteres de una cadena.

Se puede acceder a los caracteres 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
$cadena = "Pernambuco";
echo "Cadena a analizar: '$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>";
?>
```

***strlen()*.**

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

```
strlen(cadena)
```

***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)
```

***substr()*.**

Habrán 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, longitud)
```

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
$email = "maurodosantos@pernambuco.com ";
echo "Email a analizar: '$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>";
    }
}
?>
```

trim().

Elimina los espacios en blanco al principio y final de una cadena. En realidad elimina los caracteres definidos en la máscara. Por defecto: espacios, tabuladores, salto de línea y retorno de carro..

```
trim(cadena, [mascara])
```

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
$email = " maurodosantos@pernambuco.es ";
echo "Dirección recibida: '$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'.";
?>
```

***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
$email = " MAUROSOSANTOS@PERNAMBUCO.ES ";
echo "Dirección recibida: '$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
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>";
}
?>
```

***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, [sep decimal], [sep de miles])
```

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

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

Algunas más

- **explode(delimitador, string)**. Convierte en array la cadena mediante el delimitador.
- **chop(cadena) o rtrim(cadena)**. Elimina los saltos de línea y los espacios finales de una cadena.
- **ucfirst(cadena)**. Convierte el primer carácter de una cadena a mayúsculas.
- **ucwords(cadena)**. Convierte a mayúsculas el primer carácter de cada palabra de una cadena.
- **md5(cadena)**. Calcula el hash md5 de un string.
- **nl2br(cadena)**. Inserta saltos de línea HTML antes de todas las nuevas líneas de un string.

Ejemplo completo

```
<html>
<head>
  <title>Ejemplo de cadenas en PHP 7</title>
</head>
<body>
<?php
  // Strlen()
  echo "<strong>Ejemplo de Strlen()</strong>
<br>".strlen("12345")."<br>";
  echo "<hr>";

  // Explode()
  $pieza = "una-dos-tres-cuatro-cinco";
  $piezas = explode("-", $pieza);

  echo "<strong>Ejemplo de Explode()</strong> <br>";

  foreach($piezas as $individuales)
  {
    echo $individuales."<br>";
  }
  echo "<hr>";

  // Substr()
  $cadenaSubstr = "Hola mundo. Esta es una cadena a evaluar.";
  echo "<strong>Ejemplo de Substr()</strong> <br>";
  $cadenaSubstr = substr($cadenaSubstr, 4, 10);
  echo $cadenaSubstr."<br>";
  echo "<hr>";

  // Chop()
  echo "<strong>Ejemplo de Chop()</strong> <br>";
  $cadenaChop = "Hola mundo ";
  echo "<pre>";
  echo chop($cadenaChop);
  echo "</pre >";
```

```

echo "<hr>";

// Strpos()
echo "<strong>Ejemplo de Strpos()</strong> <br>";
$cadenaStrpos = 'Hola mundo. Esta es una cadena a evaluar.';
$encontrar = 'mundo';
$pos = strpos($cadenaStrpos, $encontrar);

if ($pos === false)
{
    echo "Ops! la cadena <i>$encontrar</i> no fue encontrada en la
cadena <strong>$cadenaStrpos</strong>.";
}
else
{
    echo "La cadena <i>$encontrar</i> fue encontrada en la cadena
<strong>$cadenaStrpos</strong> y existe en la posición
<strong>$pos</strong>.";
}
echo "<hr>";

// Str_replace()
echo "<strong>Ejemplo de Str_replace()</strong> <br>";

$vocales = array("a", "e", "i", "o", "A", "E", "I", "O");
$cadenaSTRreplace = "Hola mundo. Esta es una cadena a evaluar.";
$reemplazador = array("4", "3", "1", "0", "4", "3", "1", "0");

$cadenaSTRreplace = str_replace($vocales , $reemplazador,
$cadenaSTRreplace);

echo $cadenaSTRreplace."<br>";
echo "<hr>";

// Ucfirst()
echo "<strong>Ejemplo de Ucfirst()</strong> <br>";

$cadenaUCfirst = 'hola mundo';
$cadenaUCfirst = ucfirst($cadenaUCfirst);

echo $cadenaUCfirst."<br>";
echo "<hr>";

// Ucwords()
echo "<strong>Ejemplo de Ucwords()</strong> <br>";

$cadenaUcwords = 'hola mundo';
$cadenaUcwords = ucwords($cadenaUcwords);

echo $cadenaUcwords."<br>";
echo "<hr>";

// Strtolower()
echo "<strong>Ejemplo de Strtolower()</strong> <br>";

$cadenaStrtolower = 'HOLA MUNDO';
$cadenaStrtolower = strtolower($cadenaStrtolower);

echo $cadenaStrtolower."<br>";
echo "<hr>";

// Strtoupper()
echo "<strong>Ejemplo de Strtoupper()</strong> <br>";

$cadenaStrtoupper = 'hola mundo';
$cadenaStrtoupper = strtoupper($cadenaStrtoupper);

```

```
echo $cadenaStrtoupper."<br>";
echo "<hr>";

// Trim()
echo "<strong>Ejemplo de Trim()</strong> <br>";
$cadenaTrim = "    Hola mundo    ";
echo "<pre>";
echo trim($cadenaTrim);
echo "</pre >";
echo "<hr>";

// Md5()
echo "<strong>Ejemplo de md5()</strong> <br>";
$cadenaMD5 = "Cadena cualquiera";
$cadenaMD5 = md5($cadenaMD5);
echo $cadenaMD5."<br>";
echo "<hr>";

// Nl2br()
echo "<strong>Ejemplo de Nl2br()</strong> <br>";
$cadenaNl2br = "Hola\nmundo\nHTML";
echo "<pre>";

$cadenaNl2br = nl2br($cadenaNl2br);
echo $cadenaNl2br;

echo "</pre >";
echo "<hr>";

?>

</body>
</html>
```


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.

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 *índice* longitud-1.

\$capitales				
Elemento 0	Elemento 1	Elemento 2	...	Elemento n
Roma	París	Lisboa	...	Dublín

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
$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
    $países[] = "Italia"; //Añade el elemento con índice 0
    $países[] = "Francia"; //Añade el elemento con índice 1
    $países[] = "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
    $frutas = array("melón", "sandía", "naranja");
    $frutas[] = "manzana";
    $frutas[] = "melocoton";
?>
```

Existe una tercera forma de definir los arrays y es con array vacío y posteriormente se van rellenando. Además, a partir de php 5.4 se puede utilizar la notación []

Ejemplo 3.3.3.b. Aquí podéis ver la dos formas de definir un array vacío y posteriormente ir añadiendo elementos.

```
<?php
    $frutas = array();
    $frutas[] = "manzana";
    $frutas[] = "melocoton";

    $frutas2 = []; // Esta forma es LA MÁS COMUN a día de hoy
    $frutas2[] = "manzana";
    $frutas2[] = "melocoton";
?>
```

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
    $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].";
?>
```

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

Ejemplo 3.3.6. Recorriendo un array para mostrarlo por pantalla.

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

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
$astros = array("planeta", "cometa", "Venus", "Jupiter");
echo "Ordenación distinguiendo mayúsculas y minúsculas:<br>";
sort($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>";
}
?>
```

Arrays no secuenciales.

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

\$capitales

Elemento 7	Elemento 2	Elemento 19	...	Elemento 63
Roma	París	Lisboa		Dublín

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.

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.

\$capitales

Elemento "Italia"	Elemento "Francia"	Elemento "Portugal"	...	Elemento "Irlanda"
Roma	París	Lisboa		Dublín

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
$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
$nombrres = array("Eva", "Antonio", "Felipe", "Rosa");
$edades = array("Eva" => 28,
               "Antonio" => 43,
               "Felipe" => 32,
               "Rosa" => 55);
for ($i = 0; $i < count($nombrres); $i++) {
    echo "$nombrres[$i] tiene {$edades[$nombrres[$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*.

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.

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][$j] = chr(rand(65, 90));
        }
    }
    echo "SOPA DE LETRAS<br>";
    for ($i = 0; $i < ALTO; $i++) {
        for ($j = 0; $j < ANCHO; $j++) {
            echo $sopa_letras[$i][$j];
        }
        echo "<br>";
    }
}?></font>

```

Mostrar el contenido de un array.

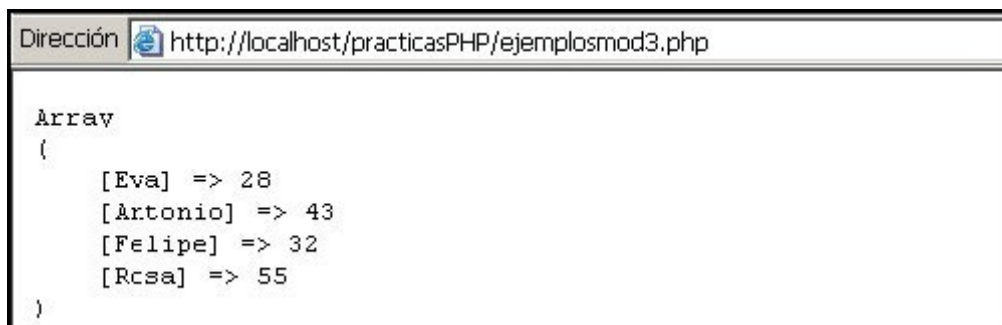
Ejemplo 3.2.15. *PHP* dispone de la función `print_r()` a la que se le pasa como argumento un array, e imprime por pantalla su contenido. Para poderlo ver bien se usa la etiqueta HTML "`<pre>... </pre>`" que sirve para mostrar el texto y los saltos de línea tal cual se ha escrito.

```

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

    print_r($edades);
?>
</pre>

```



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

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.

```
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*).

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

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

Comprobación y eliminación de posiciones de Arrays.

Es una situación muy habitual (sobretudo en arrays asociativo) tener que comprobar si una posición de un array existe, o en ocasiones eliminar la posición de un array. Las funciones que realizan esto, sólo sirven para arrays, pero sí son muy habituales en un array.

isset()

La función *isset* comprueba si una variable está definida o no en el script de PHP que se está ejecutando. Determina si una variable está definida y no es *NULL*.

Otras funciones similares son:

- [PHP is_null\(\)](#): Para comprobar si una variable es *NULL*.
- [PHP empty\(\)](#): Para saber si una variable está vacía.

Sintaxis:

```
isset ($var1, $var2, $var3...)
```

- La función devuelve **TRUE** o **FALSE**.
- **var1 (Obligatorio)**: Variable que queremos comprobar si está definida.
- **var 2, var3... (Opcionales)**: Variables adicionales a comprobar.

Ejemplos de PHP isset()

```
//Ejemplo con variable definida
$variable = 'valor';
if (isset($variable))
{
    echo "Variable definida!!!";
}
else
{
    echo "Variable NO definida!!!";
}

//Ejemplo con variables definidas
$variable = 'valor';
$variable2 = 'valor2';
if (isset($variable,$variable2))
{
    echo "Variables definidas!!!";
}
else
{
    echo "Variables NO definidas!!!";
}

//Ejemplo con variables NO definidas
$variable = 'valor';
$variable2 = 'valor2';

if (isset($variable,$variable2, $variable3))
{
    echo "Variables definidas!!!";
}
else
{
    echo "Variables NO definidas!!!";
}
```

Ejemplos de PHP isset() con arrays

```
$jugadores = array ('portero' => 'Casillas', 'Defensa' => 'Pepe');
if (isset($jugadores['portero']))
{
    echo "Variable de array definida!!!";
}
else
{
    echo "Variable de array NO definida!!!";
}

$jugadores = array ('portero' => 'Casillas', 'Defensa' => 'Pepe');
if (isset($jugadores['delantero']))
{
    echo "Variable de array definida!!!";
}
else
{
    echo "Variable de array NO definida!!!";
}
```

El uso más habitual de isset es con los array superglobales \$_GET, \$_POST y \$_REQUEST.

unset()

La función unset elimina la variable siempre y cuando esté definida. Elimina cualquier tipo de variable, array u objetos.

Sintaxis:

```
unset ($var1, $var2, $var3...)
```

Ejemplos de PHP unset()

```
$variable1 = "valor1";
$variable2 = "valor2";
$variable3 = "valor3";
var_dump ($variable1, $variable2, $variable3);

unset($variable1, $variable2, $variable3);
var_dump ($variable1, $variable2, $variable3);
//muestro el array numerico original
$array1 = array(1,2,3,4,5,6);
var_dump ($array1);

//elimino el tercer elemento del array y muestro el array
unset($array1[2]);
var_dump ($array1);

//elimino el primer y segundo elemento del array y muestro el array
unset($array1[0],$array1[1]);
var_dump ($array1);
//Declaro el array asociativo y muestro su valor
$array2 = array(unos=>1, dos=>2, tres=>3, cuatro=>4, cinco=>5, seis=>6);
var_dump ($array2);

//Elimino el elemento con clave = dos y muestro el array
unset($array2['dos']);
var_dump ($array2);

//Elimino los elementos con clave = uno y cinco, despues muestro el array
unset($array2['uno'],$array2['cinco']);
var_dump ($array2);
```

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*.

Funciones definidas por el usuario.

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
}
```

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();
?>
```

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
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.";
?>
```

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
function cuadrado($numero) {
    return $numero * $numero;
}

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

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).

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
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
function intercambiar(&$argumento1, &$argumento2) {
    $auxiliar = $argumento1;
    $argumento1 = $argumento2;
    $argumento2 = $auxiliar;
}

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

```

Alcance de las variables.

Dentro de las funciones también podemos declarar nuevas variables, pero, ¿qué 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
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
    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.

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

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.

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

```
<?php
include("config.php");
include("utils.php");
$radio = 5;
$circunferencia = 2 * $radio * PI;
$area = PI * cuadrado($radio);
echo "Un círculo de radio $radio tiene circunferencia ";
echo "$circunferencia y área $area.<br>";
$area = -8;
if (es_positivo($area)) {
    $radio = raiz($area / PI);
    echo "Un círculo de área $area tiene un radio $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.

require_once e include_once

Cuando la cantidad de includes es muy grande existe la posibilidad que hayan archivos dependientes o incluso bucles de includes

. Por ejemplo

Fichero a.php

```
include 'b.php';
```

Fichero b.php

```
include 'b.php';
```

Fichero c.php

```
include 'a.php';
```

El fichero A incluye el B y el fichero B incluye el C y el fichero C incluye el A.
Problema: Tenemos un bucle.

Otro ejemplo:

Fichero a.php

```
include 'b.php';
```

```
include 'c.php';
```

Fichero b.php

```
include 'c.php';
```

Fichero A incluye B y C

Fichero B incluye C.

Problema: Incluimos dos veces el C. Cargamos dos veces→ Error!!!!

Para evitar la carga duplicada (y los errores que ellos conlleva) y los bucles, podemos utilizar `require_once` e `include_once`.

Ambas funcionan igualmente que sus versiones sin **_once** pero al utilizar `require_once` o `include_once`, sólo se carga una vez cada fichero incluido, por tanto si detecta que quiere cargar un fichero anteriormente cargado, no lo hace y no se producen dichos errores.

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.

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:

System	Windows NT MANUEL 5.0 build 2195
Build Date	May 12 2005 12:41:39
Configure Command	cscript /nologo configure.js "--enable-snapsho
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\CursoPHP\xampp\apache\bin\php.ini
PHP API	20031224
PHP Extension	20041030
Zend Extension	220040412
Debug Build	no
Thread Safety	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, ssl, ssl3, ssl2, tls

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v2.0.4-dev, Copyright (c) 1998-2004 Zend Technologies

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.

Variables y funciones. Qué trae de nuevo PHP 7

Tipado fuerte en PHP: Descriptivo y Estricto

Como hemos visto hasta ahora, PHP es un lenguaje de programación de tipado débil, vamos que no es necesario definir el tipo de las variables, ya que lo toma automáticamente.

Esto que puede parecer una ventaja, puede conllevar código extra a la hora de manejo de tipos y ser muy cuidadoso con los cambios automáticos y no todos los programadores lo consideran una buena práctica. Por tanto, cuando PHP presentó su nueva versión añadió una serie de características y una de ellas es el chequeo de tipos en el paso de parámetros y devolución de funciones.

A partir de php 7.0, para satisfacer las necesidades de muchos programadores introduzco tipado FUERTE. Si bien existe dos tipos de tipado Fuerte en php. El tipado **fuerte descriptivo** y el tipado **fuerte estricto**.

Con el tipado fuerte, conseguimos hacer las funciones más expresivas al definirlas. Además, si se llaman con datos incorrectos el intérprete se quejará o se generará un error, dependiendo de nuestra configuración. En función del comportamiento que elija (descriptivo ó estricto).

Tipado fuerte descriptivo

En el tipado descriptivo, la diferencia fundamental es que realizará la conversión de tipos automáticamente siempre y cuando sea compatible.

Ahora veamos cómo se define una función en donde se comprueban los tipos de datos, escribiendo de nuevo la función de arriba. Queda así:

```
<?php
function primeraLetra(string $palabra):string{
    return $palabra[0];
}??
```

Ahí ya estamos indicando que recibimos una variable de tipo string, y que devolvemos igualmente un string. Cuando otro programador lea la definición de la función sabrá exactamente lo que dicha función hace.

Además Incluso si la llamamos con el número 500, PHP lo convertirá a cadena.

```
echo "La primera letra de 500 es: " . primeraLetra(500);
```

Quedando la salida de la siguiente manera

```
La primera letra de 500 es: 5
```

¿Pero que pasaría en el caso de que en vez de una función que recibe un int recibiera una cadena de texto?

```
define ('PI', 3.141591);
function calculaAreaCirculo (int $radio):float{
    return PI* pow($radio, 2);
}
//Esto funciona correctamente
echo "Área de circulo de radio: $radio es: " . calculaAreaCirculo(5);
// Esto funciona pues realiza la conversión
echo "Área de circulo de radio: $radio es: " . calculaAreaCirculo('5');
//Esto falla pues no es un tipo compatible
echo "Área de circulo de radio: $radio es: " . calculaAreaCirculo('aa');
```

La última llamada genera error, pues no sólo no es del tipo correcto, sino que además no puede realizar el casteo ("palabro" para conversión de tipos) correctamente.

Por cierto, no únicamente podemos pasar strings. También acepta el tipo de dato array, callable, bool, float e int. (Sí, hay booleanos en php)

Tipado fuerte Estricto

Aquí lo que hacemos es forzar a php para que funcione en modo estricto de manera que **NO REALICE EL CASTEO AUTOMÁTICO** de tipos.

Ejemplo:

```
declare(strict_types=1);
define ('PI', 3.141591);
function calculaAreaCirculo (int $radio):float{
    return PI* pow($radio, 2);
}
//Esto funciona correctamente el número es entero
echo "Área de circulo de radio: $radio es: " . calculaAreaCirculo(5);
// Evidentemente lo siguiente falla, pues ni si quiera es casteable
echo "Área de circulo de radio: $radio es: " . calculaAreaCirculo('aa');
//Esto falla pues aunque es convertible, es de tipo string
echo "Área de circulo de radio: $radio es: " . calculaAreaCirculo('5');
```

En el último caso se produce un error debido a que, aunque podría realizar el casting, le hemos dicho que no lo haga, y '5' no es un datos int.

Lo ideal al respecto de la declaración de tipos estricta hubiera sido poder declararla a nivel de proyecto. No obstante, dado que si definimos algo así en el php.ini, afectaría al código de

nuestras dependencias, los desarrolladores de php 7 prefirieron ser conservadores y definirlo a nivel de proyecto.

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.

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.

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>
```

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.

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).

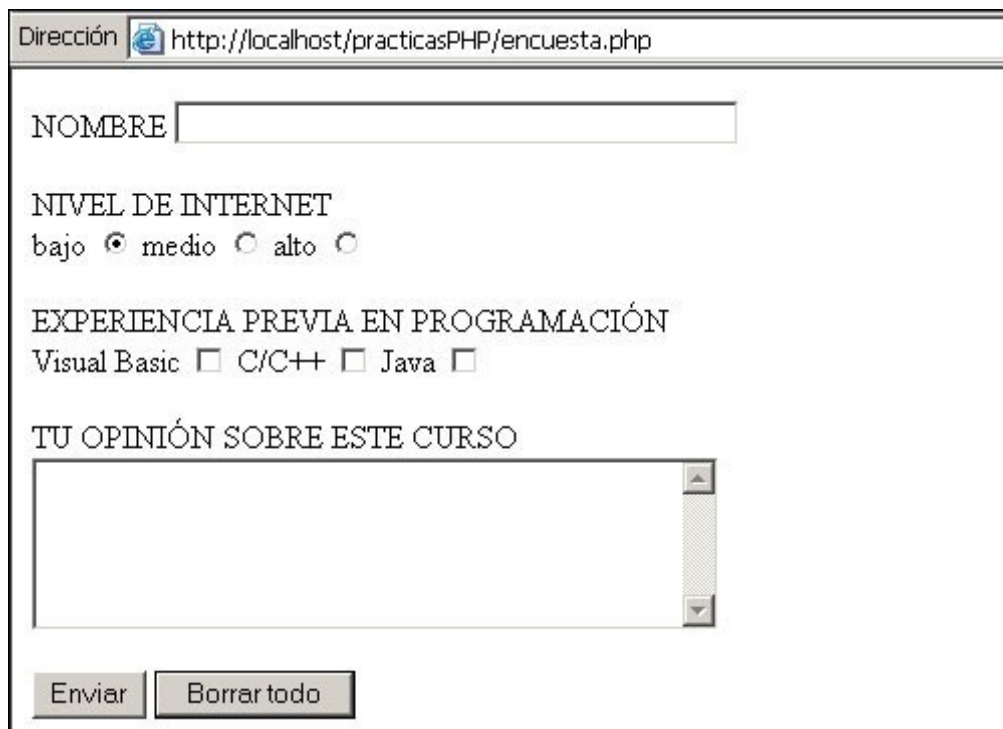
```
<input type="reset" name="Borrar todo">
```


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:



Dirección  http://localhost/practicaspHP/encuesta.php

NOMBRE

NIVEL DE INTERNET
bajo ☒ medio ☐ alto ☐

EXPERIENCIA PREVIA EN PROGRAMACIÓN
Visual Basic ☐ C/C++ ☐ Java ☐

TU OPINIÓN SOBRE ESTE CURSO

Recepción de datos de formularios.

Recibir los datos no puede ser más sencillo. El script de destino tendrá definida una 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. 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 navegador¹.

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

Ejemplo 3.4.3. 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
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>
?>
```

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 recibiéramos los datos, mostraremos el formulario.

Debemos enviar el formulario, mediante su atributo *action*, a la propia página web que contiene el formulario. Para ello, podemos emplear la variable predefinida `$_SERVER['PHP_SELF'];`, que contiene el nombre de archivo de la página web actual.

NOTA: El array superglobal `$_SERVER` es un array que contiene información, tales como cabeceras, rutas y ubicaciones de script. Las entradas de este array son creadas por el servidor web. No hay garantía que cada servidor web proporcione alguna de estas entradas. Para saber más <https://www.php.net/manual/es/reserved.variables.server.php>

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"*.

```
<?php
$opciones = array("piedra", "papel", "tijeras");
if (isset($_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
```

```

echo isset($_REQUEST["jugada"])?"Has elegido {"$_REQUEST['jugada']}, yo he
elegido $mijugada $resultado <br>¿Quieres jugar otra vez?<br>":""";
?>
<form "juego" method="post" action="<?= $_SERVER['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.

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 línea 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.

Un uso muy habitual del *header location* es realizar *get* implícitos de manera automática.

Por ejemplo:

```

$usuario='Juan';
$password='1234';
$usu=$_GET['usu'];
$pass=$_GET['pass'];

if($usuario==$usu && $password==$pass)
    header ('location:inicio.php?usuario='.$usuario);
else
    header ('location:login.php?error=Usuario o contraseña incorrectos');

```

Aquí iremos o a la página de inicio de la aplicación enviando el usuario o a la página de login donde mostramos un error.

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>
```

i

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>
```

Operador de fusión de null o Null coalesce en PHP

PHP 7 trajo muchas cosas buenas, una de ellas es el **operador de fusión null** o **null coalesce**. Este operador es representado por dos signos de interrogación. También se le conoce como operador dos signos de interrogación, ??(que original).

El uso de este operador aumenta la productividad, menos líneas y más limpias

Explicación y ejemplos de null coalesce en PHP

Este operador viene a reemplazar los if, isset y tal vez los operadores ternarios. Lo que hace este operador es devolver el primer valor que no sea nulo.

Pongamos el ejemplo básico en el que nos mandan por medio de GET el límite de usuarios que queremos ver.

Si no mandan el límite, entonces asumimos un límite por defecto, que sería 10 (en este caso). Entonces lo haríamos así:

```
if(isset($_GET["limite"])){
    $limite = $_GET["limite"];
}else{
    $limite = 10;
}

hacerConsultaDeUsuariosConLimite($limite);
```

Si queremos ser un poco más elegantes, usamos el operador ternario y quedaría rescrito así:

```
$limite = isset($_GET["limite"]) ? $_GET["limite"] : 10;
hacerConsultaDeUsuariosConLimite($limite);
```

Pero igual no me gusta, mejor lo hacemos así:

```
$limite = $_GET["limite"] ?? 10;
hacerConsultaDeUsuariosConLimite($limite);
```

Así se ve algo mejor. En este momento no se nota mucho la diferencia, pero sigue leyendo para que veas cuán útil es.

Más de dos operandos

Siguiendo el ejemplo de arriba, ahora supongamos que modificamos nuestra API y aceptamos el límite tanto por POST así como por GET.

Normalmente quedaría así:

```
if(isset($_GET["limite"])){
    $limite = $_GET["limite"];
}else if(isset($_POST["limite"])){
    $limite = $_POST["limite"];
}else{
    $limite = 10;
}

hacerConsultaDeUsuariosConLimite($limite);
```

Con el operador ternario, quedaría así:

```
$limite = isset($_GET["limite"]) ? $_GET["limite"] :
isset($_POST["limite"]) ? $_POST["limite"] : 10;
hacerConsultaDeUsuariosConLimite($limite);
```

Finalmente, con este nuevo operador queda de la siguiente manera:

```
$limite = $_GET["limite"] ?? $_POST["limite"] ?? 10;
hacerConsultaDeUsuariosConLimite($limite);
```

Primero evalúa si hay algo en GET, luego en POST y finalmente, si ambos valores son nulos, se obtiene 10.

NOTA: Ciertamente se podría usar el `$_REQUEST` y en vez del código anterior podríamos hacer lo siguiente

```
$limite = isset($_REQUEST["limite"]) ? $_REQUEST["limite"] : 10;
hacerConsultaDeUsuariosConLimite($limite);
```

Pero el ejemplo de GET y POST es para ilustrar el funcionamiento del Operador coalesce null.

3, 4, 5 o más operandos

Se puede evaluar una lista infinita de operandos. Aquí un ejemplo:

```
$limite = null ?? null ?? null ?? 10 ?? 20 ?? 30;
```

En este caso, límite es 10, ya que es **el primer valor que no es nulo** y ahí deja de evaluar:

También trabaja bien con funciones:

```
function hola(){
    return "Hola mundo";
}

function adios(){
    return null;
}

$saludo = adios() ?? hola();
En este caso, saludo es "Hola mundo":
```

Así como **podemos llamar a funciones** creadas por nosotros, podemos llamar a funciones nativas de PHP y claramente pasar argumentos.

Ejemplo final

Siguiendo nuestros ejemplos de usuarios en donde pueden mandar el límite por GET o POST.

Tendremos una función que devolverá los usuarios, pero que también puede devolver null. Y si devuelve null, devolvemos un array vacío. Juntando todo...

```
hacerConsultaDeUsuariosConLimite($_GET["limite"] ??
    $_POST["limite"] ?? 10) ?? [];
```

Llamamos a la función ya sea con lo que haya en GET, POST o con el número 10. Y si nos devuelve null, entonces devolvemos []; es decir, un array vacío.

Básicamente este operador simplifica el uso de `isset` y el encadenamiento largo con el operador ternario. Devuelve el primer argumento que no sea nulo, y podemos evaluar una lista infinita de operandos.

Un poco de Seguridad en el Envío de Datos. Sanitización de datos. Limpiar y validar variables GET y POST en PHP

Limpiar y validar variables GET. Si estás trabajando en un sistema seguramente se han encontrados con formularios que trabajan con métodos GET y POST para manejar la información del sistema o quizá al momento de editar un registro usan el identificador del registro para pasarlo a la URL y mostrar tal registro.

En estos casos es de vital importancia limpiar las variables de posibles inyecciones SQL que pudiéramos ser víctimas y posterior evitarnos dolores de cabeza ante accesos no autorizados en nuestra plataforma.

¿Cuál es la solución para estos problemas?

El método es muy sencillo de aplicar y solo se debe agregar al inicio de nuestros ficheros “.php” que se van a usar en el proceso. Sin embargo, el trabajo conciso de este método es recorrer todos los valores recibidos por [GET](#) y [POST](#) y limpiarlos de cadenas peligrosas.

Validación y desinfección de datos con filtros

Limpiar y validar la entrada del usuario es una de las tareas más comunes en una aplicación web. Para facilitar esta tarea, [PHP](#) proporciona una extensión de filtro nativa que puede utilizar para desinfectar o validar datos como direcciones de correo electrónico, direcciones URL, direcciones IP, números, cadenas de texto, etc.

Para validar los datos utilizando la extensión de filtro es necesario utilizar la función **filter_var()** de PHP. La sintaxis básica de esta función se puede dar con:

Sintaxis

`filter_var(variable, filter, options)`

Esta función toma tres parámetros de los cuales los dos últimos son opcionales. El primer parámetro es el valor que se va a filtrar, el segundo parámetro es el identificador del filtro que se va a aplicar y el tercer parámetro es la matriz de opciones relacionadas con el filtro. Veamos cómo funciona.

En el ejemplo siguiente se desinfectará una cadena quitando todas las etiquetas HTML de ella:

a) Ejemplo de limpiar y validar texto

```
<?php
$comentario = "<h1>Hola! Buenos dias con todos</h1>";

// Desinfectar e imprimir cadena de comentarios
$limpiacomentario= filter_var($comentario , FILTER_SANITIZE_STRING);
echo $limpiacomentario;
?>
```

La salida del ejemplo anterior se verá algo como esto:

```
Hola! Buenos días con todos
```


b) Limpiar y validar valores enteros

En el ejemplo siguiente se valida si el valor es un entero válido o no.

```
<?php
// Ejemplo de variable numerica no valida
$novalido = "//25[]";

// Quitamos los caracteres ilegales de la variable
$numero = filter_var($novalido, FILTER_SANITIZE_NUMBER_INT);

// Validamos la variable filtrada
if(filter_var($numero, FILTER_VALIDATE_INT)){
echo "EL $numero ES NUMERO VALIDO";
} else{
echo "EL $numero NO ES NUMERO VALIDO";
}
?>
```

La salida en el navegador:

EL 25 ES NUMERO VALIDO

Lista de filtros de sanitización

La lista completa la podemos obtener de la siguiente página

<https://www.php.net/manual/es/filter.filters.sanitize.php>

Alguno ejemplos que podríamos usar son:

FILTER_SANITIZE_EMAIL.- Elimina todos los caracteres menos letras, dígitos y `!#$%&*+.-=?^_`{|}~@.[]`.

FILTER_SANITIZE_ENCODED.- String URL-encode, opcionalmente elimina o codifica caracteres especiales.

FILTER_SANITIZE_NUMBER_INT.- Elimina todos los caracteres excepto dígitos y los signos de suma y resta.

FILTER_SANITIZE_STRING.- Elimina etiquetas, opcionalmente elimina o codifica caracteres especiales.

Vamos a lo práctico: filter_var_array

El **filter_var_array** me permite de una manera más cómoda asignar los distintos filtros a los datos que obtengo después del envío de los mismos: Veamos un ejemplo simple

```
<form action="prueba.php" method="post">
Numero: <input type="text" name="numero"><br>
Email: <input type="text" name="email"><br>
texto: <input type="text" name="texto"><br>
Enviar: <input type="submit" name="enviar"/>
</form>
<?php
if(isset($_POST['enviar'])) {
    $_POST = filter_var_array($_POST, array(
        "numero" => FILTER_SANITIZE_NUMBER_INT ,
        "email" => FILTER_SANITIZE_EMAIL,
        "texto" => FILTER_SANITIZE_STRING,
    ));
    var_dump($_POST);
}
?>
```

Con esto datos de entrada:

46

Numero:
Email:
texto:
Enviar:

Obtendremos:

C:\xampp\htdocs\practicas\prueba.php:15:

```
array (size=3)
  'numero' => string '34' (length=2)
  'email' => string 'viv#{}--@gmail.com' (length=18)
  'texto' => string 'dfdfd' (length=5)
```

Como puedes ver han “cambiado” los datos siguiendo el criterio de los filtros.

La ventaja del `filter_var_array` es evidente, en vez de asignar, uno por uno los distintos `filter_var` a cada variable, puedo realizar la asignación de los filtros usando un array de filtros. Eso sí, si se produjera un error el método devolvería `false`

Con esto conseguimos limitar los ataques de Cross Site Scripting, file upload, file inclusión, CSRF... Esto es debido a que “**negamos**” los caracteres que permiten la ejecución de los comandos que se necesitan para dichos ataques. ¿Cómo de seguro es esto? Si bien la seguridad absoluta no existe, la sanitización de datos es una herramienta fundamental y clave para mejorar el nivel de seguridad de nuestras aplicaciones.

Otro ejemplo más completo:

```
<?php
error_reporting(E_ALL | E_STRICT);
$data = array(
    'id_producto' => 'libgd<script>',
    'componente' => '10',
    'versiones' => '2.0.33',
    'test_escalar' => array('2', '23', '10', '12'),
    'test_array' => '2',
);

$args = array(
    'id_producto' => FILTER_SANITIZE_ENCODED,
    'componente' => array('filter' => FILTER_VALIDATE_INT,
        'flags' => FILTER_FORCE_ARRAY,
        'options' => array('min_range' => 1,
            'max_range' => 10)
    ),
    'versiones' => FILTER_SANITIZE_ENCODED,
    'no_existe' => FILTER_VALIDATE_INT,
    'test_scalar' => array(
        'filter' => FILTER_VALIDATE_INT,
        'flags' => FILTER_REQUIRE_SCALAR,
    ),
    'test_array' => array(
        'filter' => FILTER_VALIDATE_INT,
        'flags' => FILTER_FORCE_ARRAY,
    )
);

$myinputs = filter_var_array($data, $args);

var_dump($myinputs);
echo "\n";
```

El resultado del ejemplo sería:

C:\xampp\htdocs\practicas\prueba.php:49:

```
array (size=6)
  'id_producto' => string 'libgd%3Cscript%3E' (length=17)
  'componente' =>
    array (size=1)
      0 => int 10
  'versiones' => string '2.0.33' (length=6)
  'no_existe' => null
  'test_scalar' => null
  'test_array' =>
    array (size=1)
      0 => int 2
```

Observa que tenemos algunas opciones más:

```
'componente' => array('filter' => FILTER_VALIDATE_INT,
                      'flags' => FILTER_FORCE_ARRAY,
                      'options' => array('min_range' => 1,
'max_range' => 10))
```

Vemos que el element 'componente' se le asignan 3 opciones.

1. **filter**: Filtro aplicar. Ya conocido
2. **flags**: El resultado del filtro lo devuelve en un array.
3. **options**: Rango de valores

Existen muchas más opciones pero con esto tenemos más que suficiente para empezar.

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.

Manejo de ficheros.

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.
a+	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).

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

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.

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

En realidad existen multitud de opciones de lectura. Aquí van más

1. *fgets*

```
string fgets (resource $handle [, int $length ])
```

Obtiene una línea desde el puntero de un archivo. La lectura termina cuando se llegue a *\$length*, se llegue a una nueva línea o se alcance el final del archivo.

```
$fp = fopen("miarchivo.txt", "r");
while (!feof($fp)){
    $linea = fgets($fp);
    echo $linea;
}
fclose($fp);
```

2. *fread*

```
string fread (resource $handle, int $length)
```

Lectura de un fichero en modo binario. Se puede incluir un parámetro opcional *\$length* para limitar lo que devuelva.

Esta función tiene una función diferente a *fgets()*, ya que lee archivos binarios (archivos que no están creados para ser legibles por humanos). Ya que las líneas no son algo muy común en este tipo de archivos, se suele especificar el tamaño *\$length* en bytes a devolver:

```
$fp = fopen ("miarchivo.txt", "rb");
$datos = fread($fp, 4096);
//Es frecuente emplear la función filesize() en lugar de un número
// concreto para especificar que se quiere devolver el archivo entero:
$fp = fopen("miarchivo.txt", "rb");
$datos = fread($fp, filesize("miarchivo.txt"));
fclose($fp);
```

3. fscanf

```
mixed fscanf (resource $handle, string $format [, mixed &$... ])
```

Interpreta la entrada de un archivo *\$handle* en función de un formato *\$format*. El formato es el mismo que se emplea en la función *sprintf()*. Cada llamada a *fscanf()* lee una línea del archivo. Cualquier espacio en blanco en la cadena de formato coincide con cualquier espacio en blanco en el flujo de entrada. Esto significa que incluso una tabulación *\t* en la cadena de formato puede coincidir con un simple carácter espacio en el flujo de entrada.

```
$fp = fopen("miarchivo.txt", "r");
while ($animalinfo = fscanf($fp, "%s\t%s")){
    list($animal, $nombre) = $animalinfo;
    echo "El " . $animalinfo[0] . " se llama {$animalinfo[1]} <br>";
//Seria lo mismo que la línea anterior lo siguiente
    echo "El $animal se llama $nombre <br>";
}
fclose($fp);
```

Observa que *fscanf*, lleva una serie de parámetros *%s*.

Si bien la lista es más larga y compleja, nos basta con saber los siguientes:

Parámetro	Tipo de Datos
%s	String, cadena texto
%d	Entero.
%f	Número real.

4. fgetc

```
string fgetc (resource $handle)
```

Obtiene sólo un carácter desde un puntero de un archivo. Devuelve el string con el único carácter. Devuelve false si es EOF..

```
if($fp = fopen('miarchivo.txt', 'r')) {
    while (false !== ($character = fgetc($fp))) {
        echo "$character\n";
    }
}
```

5. file_get_contents

```
string file_get_contents (string $filename [, bool $use_include_path =
false [, resource $context [, int $offset = -1 [, int $maxlen ]]] )
```

Devuelve un archivo entero a una cadena, se puede especificar el comienzo desde *\$offset* hasta *\$maxlen*.

Si falla devuelve false, pero genera un *EWARNING* si *\$filename* no se pudo encontrar, *\$maxlen* es menor que cero o si falla la búsqueda del *\$offset* especificado en el flujo. Se utiliza mucho para leer web ajenas.

```
// Devuelve el contenido de una web
$web = file_get_contents("http://www.ejemplo.com");
echo $web;
```

52

```
// Devuelve contenido de un archivo en el directorio actual
$contenido = file_get_contents("miarchivo.txt");
echo $contenido;
// Busca dentro de include_path
$archivo = file_get_contents('./miarchivo.txt', FILE_USE_INCLUDE_PATH);
// Devuelve 20 caracteres desde el carácter 100
$articulo = file_get_contents('archivo.txt', NULL, NULL, 100, 20);
echo $articulo;
```

6. readfile

```
int readfile (string $filename [, bool $use_include_path = false [,
resource $context ]])
```

Es una versión más simplificada de `_file_getcontents()`. Lee un archivo y lo escribe en el búfer de salida. La función devuelve el número de bytes leídos del archivo.

```
// Devuelve el contenido de un archivo
readfile("miarchivo.txt");
// Devuelve el contenido y el número de bytes, 79
$contenido = readfile("miarchivo.txt");
echo $contenido;
```

7. file

```
array file (string $filename [, int $flags = 0 [, resource $context ]])
```

Transfiere **un archivo a un array**. Función similar a `_file_getcontents()` pero devuelve los contenidos en un array, donde cada elemento es una línea del archivo. El parámetro opcional `flags` puede ser una de las siguientes constantes:

- **FILE_USE_INCLUDE_PATH**. Busca el fichero en `include_path`.
- **FILE_IGNORE_NEW_LINES**. No se añade una nueva línea al final de cada elemento del array.
- **FILE_SKIP_EMPTY_LINES**. Salta las líneas vacías.

```
// Devuelve el contenido en un array:
$array = file("miarchivo.txt", FILE_IGNORE_NEW_LINES);
var_dump($array);
/*
array (size=4)
  0 => string 'Esta es la línea 1' (length=19)
  1 => string 'Esta es la línea 2' (length=19)
  2 => string 'Esta es la línea 3' (length=19)
  3 => string 'Esta es la línea 4' (length=19)
*/
```


8. Lectura desde una posición exacta

Podemos llegar exactamente a un punto del archivo con **fseek()**. Una vez situados en una posición, podemos emplear cualquiera de las funciones anteriores como *fgets()* o *fscanf()* para leer los datos:

```
$fp = fopen("miarchivo.txt", "r");
fseek($fp, 28);

while(!feof($fp)){
    $linea = fgets($fp);
    echo $linea;
}
fclose($fp);
/*
la línea 2
Esta es la línea 3
Esta es la línea 4
*/
```

Escribir.

Otra acción que queremos hacer habitualmente es añadir datos a un archivo. Para ello se usa una función muy sencilla *fwrite()* o también podemos usar *fputs()*, que escriben 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.

fwrite()

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
    $archivo = fopen("refranes.txt", "a");
    fwrite($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.

fputs()

La función `fputs()` escribe una línea en un archivo. Su sintaxis general es:

```
<?php //Ejemplo aprenderaprogramar.com
fputs(descriptorDelFichero, cadena);
?>
```

Ejemplo

```
<?php // Ejemplo aprenderaprogramar.com
// Escribimos una primera línea en fichero.txt
// fichero.txt tienen que estar en la misma carpeta que el
fichero php
$fp = fopen("fichero.txt", "w");
fputs($fp, "Prueba de escritura aprenderaprogramar.com");
fclose($fp);
?>
```

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.

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.";
}
?>
```

Copiar y Renombar un fichero.***copy(origen,destino):***

Realiza una copia del fichero origen a destino.

rename(nombreViejo, nombreNuevo)

Renombra o mueve de sitio un fichero.

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.

file_exists(ruta/nombre).

Devuelve TRUE si el archivo por el que preguntamos existe.

file_size(ruta/nombre).

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

is_file(ruta/nombre).

Devuelve TRUE si el archivo es un fichero.

is_dir(ruta/nombre).

Devuelve TRUE si el archivo es un directorio.

is_readable(ruta/nombre).

Devuelve TRUE si el archivo se puede abrir para lectura.

is_writeable(ruta/nombre).

Devuelve TRUE si el archivo se puede abrir para escritura.

is_writeable(ruta/nombre).

Devuelve TRUE si el archivo se puede abrir para escritura.

filemtime(ruta)

Devuelve la fecha y hora de la última modificación de un fichero.

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_writeable($ruta)) {
    echo "$ruta se puede abrir para escritura.<br>";
}
} else {
    echo "$ruta no existe.";
}
?>
```

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.

Las principales funciones para trabajar con directorios son:

- **opendir(ruta directorio):** abre un directorio y nos devuelve un identificador de directorio, el cual usaremos con la para su posterior procesamiento con las funciones de manejo de directorios `closedir()`, `readdir()` y `rewinddir()`.
- **readdir(id_directorio):** devuelve el nombre del siguiente fichero en un directorio. El manejador se obtiene usando anteriormente `opendir()`.
- **chdir(ruta directorio):** cambia de directorio.
- **closedir(id_directorio):** cierra un directorio.
- **is_dir(ruta directorio):** devuelve true si es un directorio y false en caso contrario.
- **mkdir(ruta directorio):** crea un directorio nuevo.
- **rename(nombreViejo, nombreNuevo):** renombra o mueve de sitio un directorio.
- **rewinddir(id_directorio):** sitúa el manejador de directorio al principio del directorio.
- **rmdir(directorio):** elimina un 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>";
        }
    }
?>
```

Ejemplo 3.5.8.b. El siguiente código muestra en una página web el contenido de un directorio, tal como lo hacen muchos servidores web. Para cada fichero se muestra su nombre, la fecha de la última modificación y el tamaño que ocupa.

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Prueba de lectura de directorio</title>
</head>
<body>
<?php
$nomdir = './';
echo "<h1>Contenido de $nomdir</h1>\n";
$dir = opendir($nomdir);

echo "<pre>\n";
echo "<b>";
echo str_pad("Nombre", 30); //str pad formatea a tamaño fijo de 30
echo str_pad("Fecha ult. mod.", 20);
echo str_pad("Tamaño", 10);
echo "</b></pre>\n";
echo "<hr /><pre>\n";

while(($fichero = readdir($dir)) != FALSE)
{
    echo str_pad($fichero, 30);
    echo str_pad(date("d/m/Y H:i" , filemtime( $fichero)), 20);
    if(is_dir( $fichero))
    {
        echo "-";
    }
    else
    {
        echo str_pad(filesize( $fichero), 10);
    }
    echo "<br />\n";
}
closedir($dir);

echo "</pre><hr />\n";
?>
</body>
</html>
```

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
$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.

Array Superglobal \$_FILES

El array superglobal \$_FILES se crea en el servidor al subir uno o varios archivos. En él, disponemos información del fichero o ficheros subidos. El contenido de los atributos de cada archivo es el siguiente:

- **name:** Nombre original del fichero.
- **type:** tipo MIME del archivo subido. Para ver todos los MIME disponibles, pincha aquí: <https://www.htmlquick.com/es/reference/mime-types.html>
- **tmp:** El servidor le da un nombre temporal al fichero subido. Este es su nombre.
- **error:** Si se ha subido correctamente o no
- **size:** Tamaño del fichero subido

Por supuesto si se suben varios ficheros cada elemento genera una posición en array \$_FILES para cada fichero cada uno con estos campos.

NOTA: El manejo del envío de ficheros se puede configurar en el fichero php.ini con las directivas file_uploads, upload_max_filesize, upload_tmp_dir, post_max_size y max_input_time.

Subida de múltiples ficheros

Para subir múltiples ficheros tenemos que modificar un formulario para que queda de la siguiente manera:

```
<form action="index.php" method="post" enctype="multipart/form-data">
  <input name="upload[]" type="file" multiple="multiple" />
</form>
```

Puntos importantes: el **enctype="multipart/form-data"** en la etiqueta *form* para que este envíe correctamente los archivos que subimos, el definir el nombre del input como un array poniendo los **corchetes []** y la etiqueta **multiple="multiple"** que nos permitirá seleccionar varios ítems a las vez.

Vamos ahora con el **PHP** con explicación línea a línea:

```
// RECORREMOS LOS FICHEROS
if(isset($_FILES)){
  for($i=0; $i<count($_FILES['upload']['name']); $i++) {
    //Obtenemos la ruta temporal del fichero
    $fichTemporal = $_FILES['upload']['tmp_name'][$i];

    //Definimos una ruta definitiva para guardarlo
    $destino = "./nuestraCarpeta/" . $_FILES['upload']['name'][$i];

    //Movemos a la ruta final
    if(move_uploaded_file($fichTemporal, $destino)) {
      //imprimimos el nombre del archivo subido
      echo"Se ha subido el fichero " . $_FILES['upload']['name'][$i];
    }
  }
}
```

Más ejemplos

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Prueba de subir fichero</title>
</head>
<body>
<form action="subirfichero.php" method="post" enctype="multipart/form-
data">
Fichero: <input type="file" name="fichero" />
<br />
<input type="submit" value="Enviar" />
</form>
</body>
</html>
```

El segundo fragmento de código es la página PHP que recibe el fichero enviado en el servidor. En este código se emplea la función `file_exists(nombre)` para comprobar si ya existe el fichero que se intenta subir y la función `move_uploaded_file(origen, destino)` para mover el fichero temporal de la subida a su destino definitivo.

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="utf-8" />
<title>Prueba de subir fichero</title>
</head>
<body>
<p>
<?php
$msgError = array(0 => "No hay error, el fichero se subió con éxito",
1 => "El tamaño del fichero supera la directiva
upload_max_filesize el php.ini",
2 => "El tamaño del fichero supera la directiva
MAX_FILE_SIZE especificada en el formulario HTML",
3 => "El fichero fue parcialmente subido",
4 => "No se ha subido un fichero",
6 => "No existe un directorio temporal",
7 => "Fallo al escribir el fichero al disco",
8 => "La subida del fichero fue detenida por la
extensión");

if($_FILES["fichero"]["error"] > 0)
{
echo "Error: " . $msgError[$_FILES["fichero"]["error"]] . "<br />";
}
else
{
echo "Nombre original: " . $_FILES["fichero"]["name"] . "<br />";
echo "Tipo: " . $_FILES["fichero"]["type"] . "<br />";
echo "Tamaño: " . ceil($_FILES["fichero"]["size"] / 1024) . " Kb<br /
>";
echo "Nombre temporal: " . $_FILES["fichero"]["tmp_name"] . "<br />";

if(file_exists("upload/" . $_FILES["fichero"]["name"]))
{
echo $_FILES["fichero"]["name"] . " ya existe";
}
else
{
move_uploaded_file($_FILES["fichero"]["tmp_name"],
```


61

```
        "upload/" . $_FILES["fichero"]["name"]);  
    echo "Almacenado en: " . "upload/" . $_FILES["fichero"]["name"];  
    }  
}  
?>  
</p>  
</body>  
</html>
```