

# Programación en PHP

---

Este documento profundiza en los principios básicos del lenguaje de programación PHP. Veremos las sentencias de control, los arrays, funciones e introduciremos la programación web mediante la gestión de los formularios.

---

## **Programación en PHP**

Copyright © 2024 by Rafael Lozano Luján.

Este documento está sujeto a derechos de autor. Todos los derechos están reservados por el Autor de la obra, ya sea en su totalidad o en parte de ella, específicamente los derechos de:

- La reproducción total o parcial mediante fotocopia, escaneo o descarga.
- La distribución o publicación de copias a terceros.
- La transformación mediante la modificación, traducción o adaptación que altere la forma de la obra hasta obtener una diferente a la original.

## Tabla de contenido

1 Sentencias.....	1
1.1 Sentencia secuencial.....	2
1.2 Estructuras de control.....	2
1.2.1 Estructura condicional simple.....	2
1.2.2 Estructura condicional doble.....	3
1.2.3 Instrucción if .. else anidada.....	4
1.2.4 Operador ?:.....	6
1.2.5 Operador de fusión de null.....	7
1.2.6 Selectiva múltiple. Sentencia switch.....	8
1.2.7 Sentencia match.....	11
1.3 Bucles.....	12
1.3.1 For.....	12
1.3.2 While.....	16
1.3.3 Do .. While.....	16
1.3.4 Romper el flujo de control. Instrucciones break y continue.....	17
1.4 Sintaxis alternativas de las estructuras de control.....	19
2 Arrays.....	19
2.1 Arrays escalares.....	20
2.2 Arrays asociativos.....	21
2.3 Arrays mixtos.....	22
2.4 Arrays bidimensionales.....	22
2.5 Arrays multidimensionales.....	24
2.6 El bucle foreach.....	24
3 Funciones.....	26
3.1 Definición de una función.....	27
3.2 Ejecución de la función.....	29
3.3 Parámetros y argumentos.....	29
3.3.1 Paso de parámetros por referencia.....	30
3.3.2 Valores parámetros por defecto.....	31
3.3.3 Parámetros con tipo de datos.....	31
3.3.4 Lista de parámetros de longitud variable.....	32
3.3.5 Argumentos con nombre.....	33
3.4 Devolución de valores.....	33
3.5 Ámbito de las variables.....	36
3.5.1 La palabra clave global.....	36
3.5.2 Variables static.....	37
3.5.3 Resumen del ámbito de variables.....	38
3.6 Definición de la función con código HTML.....	39
3.7 Recursividad.....	40
3.8 Funciones internas.....	41
4 Ficheros externos.....	42
4.1 La función include() y require().....	42
4.2 Funciones include_once() y require_once().....	43
5 Proceso de formularios.....	43
5.1 El protocolo HTTP.....	44

5.1.1 Conexión HTTP.....	44
5.1.2 Formato del mensaje HTTP.....	47
5.1.3 Los encabezados.....	48
5.2 Arrays globales EGPCS.....	49
5.3 Información del servidor.....	49
5.4 Métodos HTTP.....	50
5.5 Parámetros del formulario.....	51
5.6 Páginas autoprocesadas.....	56
5.7 Sticky forms.....	59
5.8 Parámetros multivaluados.....	61
5.9 Parámetros multivaluados sticky.....	63
5.10 Subida de archivos.....	64
5.11 Saneamiento de valores de parámetros.....	67
5.12 Validar el formato de los valores de parámetros.....	71
5.13 Validación de valores de parámetros.....	74
6 Bibliografía.....	77

# Programación en PHP

## 1 Sentencias

---

Una instrucción o sentencia es la unidad mínima de ejecución de un programa y que realiza una tarea sencilla. Un programa se compone por tanto de conjunto de instrucciones que acaban resolviendo un problema. En PHP, toda instrucción termina en punto y coma.

Un bloque de código es un conjunto de instrucciones que se comportan como una unidad. Un bloque de código está limitado por las llaves de apertura { y cierre }. Hay muchas estructuras de programación en el lenguaje PHP que emplean bloques de código, como las clases, estructuras de control, manejo de excepciones, etc.

En un bloque de código podemos tener instrucciones simples y también otro bloque de código. El cuál a su vez puede contener lo mismo. No hay límite en cuanto al anidamiento de los bloques de código.

Como en cualquier otro lenguaje de programación, la legibilidad del código es fundamental a la hora de depurarlo y comprender cómo funciona el programa. Por tanto, el sangrado de las instrucciones dentro de un bloque de código ayudará a mostrar qué instrucciones pertenecen a un bloque o a otro.

Si el bloque de instrucciones está constituido por una única instrucción no es obligatorio el uso de las llaves de apertura y cierre, aunque es recomendable utilizarlas.

En los siguientes apartados vamos a describir las instrucciones más habituales en PHP.

## 1.1 Sentencia secuencial

Las instrucciones secuenciales realizan una tarea simple. Su orden de ejecución es secuencial, una detrás de otra tal y como aparecen escritas en el código del programa.

Una instrucción secuencial puede ser: enviar un mensaje en la salida, crear una nueva variable, invocar una función o método, etc. El carácter `;` separa una instrucción de la siguiente. Al escribir código se pone una instrucción por línea, aunque instrucciones cortas pueden colocarse en una misma línea, cada una con su correspondiente `;` de separación con la siguiente instrucción.

## 1.2 Estructuras de control

Una estructura de control es aquella que modifica el flujo natural de ejecución de instrucciones secuenciales en un programa. Generalmente alteran el flujo de ejecución en función del resultado de una expresión relacional o lógica. Dependiendo de la forma en que realizan esta modificación de ejecución de instrucciones tenemos varios tipos de estructuras de control.

### 1.2.1 Estructura condicional simple

Una estructura condicional simple `if` ejecuta una instrucción o bloque de instrucciones si una condición se cumple. Su sintaxis es

```
if( <expresión> ) <instrucción>;
```

Donde

- ✓ `<expresión>` → Expresión relacional o lógica a evaluar.
- ✓ `<instrucción>` → Instrucción o bloque de código a ejecutar si `<expresión>` es `true`.

Si la expresión relacional o lógica que se escribe entre paréntesis es `true`, entonces se ejecuta la instrucción asociada a la estructura `if`. Si es `false`, el flujo de ejecución de instrucciones continua en la siguiente instrucción al `if`. Por ejemplo

```
<?php
$nota = 8.5;
if ($nota >= 5) { //inicio de la condición
    echo "Enhorabuena!!"
    echo "Has aprobado");
} // fin de la condición

echo "Fin de la estructura if";

?>
```

En el código anterior si la `$nota` es igual o superior a 5, entonces se visualizarán los mensajes. Si por el contrario, la `$nota` es inferior a 5 el resultado de la expresión es `false` y la ejecución seguirá por la siguiente instrucción al `if`.

En este ejemplo la instrucción de la estructura `if` es un bloque de código ya que queremos ejecutar más de una instrucción si la condición es verdadera.

Las sentencias `if` pueden anidarse dentro de otra sentencias `if` sin límite, lo cual provee completa flexibilidad para la ejecución condicional de diferentes partes del programa.

### 1.2.2 Estructura condicional doble

Con frecuencia se desea ejecutar una sentencia si una determinada condición se cumple y una sentencia diferente si la condición no se cumple. Esto es para lo que sirve `else`. El `else` extiende una sentencia `if` para ejecutar una sentencia en caso que la expresión en la sentencia `if` se evalúe como `false`.

La estructura `if .. else` tiene la siguiente sintaxis:

```
if( <expresión> )
    <instrucción 1>;
else
    <instrucción 2>;
```

Donde

- ✓ `<expresión>` → Expresión relacional o lógica a evaluar.
- ✓ `<instrucción 1>` → Instrucción o bloque de código a ejecutar si `<expresión>` es `true`.
- ✓ `<instrucción 2>` → Instrucción o bloque de código a ejecutar si `<expresión>` es **false**.

Si la expresión relacional o lógica que se escribe entre paréntesis es `true`, entonces se ejecuta la instrucción asociada al estructura `if`. Si es `false`, se ejecuta la instrucción asociada a la cláusula `else`. Una vez ejecuta la instrucción correspondiente, el flujo del programa continua en la siguiente instrucción después de la estructura `if`.

Podemos modificar el ejemplo anterior añadiendo una cláusula `else`.

```
<?php
$nota = 8.5;
if ($nota >= 5) { //inicio de la condición
    echo "Enhorabuena!!"
    echo "Has aprobado");
}
else {
    echo "Has suspendido";
    echo "Tienes que ir a la recuperación";
}

// fin de la condición
```

```
echo "Fin de la estructura if";
```

En el código anterior si la `$nota` es igual o superior a 5, entonces se visualizarán los primeros mensajes. Si por el contrario, la `$nota` es inferior a 5 el resultado de la expresión es `false` y se ejecutarán los mensajes asociados a la cláusula `else`. Al finalizar la ejecución de uno de los dos bloques de código, la ejecución seguirá por la siguiente instrucción al `if`.

Un bloque PHP puede cerrarse antes de cerrar una sentencia `if`. Esto nos da flexibilidad para enviar a la salida código HTML dependiendo del resultado de una expresión lógica. Veamos el siguiente ejemplo.

```
<?php
if ( $nota >= 5 ) ?>
    <p>Enhorabuena!!<br>Has aprobado</p>
<?php else ?>
    <p>Has suspendido!!<br>Tienes que ir a la recuperación</p>
<?php
echo "Fin de la estructura if";
?>
```

### 1.2.3 Instrucción `if .. else` anidada

En ocasiones, en la instrucción de un `if` o un `else` hay que incluir otra instrucción `if .. else`. Un `if` dentro de otro se denomina `if` anidado, y no hay límite de profundidad a la hora de anidar instrucciones `if`.

Un programa puede evaluar varios casos colocando instrucciones `if...else` dentro de otras instrucciones `if .. else`, para crear instrucciones `if...else` anidadas. Por ejemplo, el siguiente fragmento de programa evalúa una nota numérica para visualizar su correspondiente calificación.

```
<?php
// Obtenemos la nota desde un formulario
$nota = $_POST['nota'];

if ( $nota >= 0 and $nota < 5 ) {
    // Se ha suspendido
    echo "¡¡Suspendido!!<br>";
    echo "¡¡A recuperar!!";
}
else {
    echo "Enhorabuena!!<br>";
    echo "Has aprobado";

    // Primer if anidado
    if( $nota < 6 ) {
        echo "Tienes SUFICIENTE";
    }
    else {

        // Segundo if anidado
```

```

if( $nota < 7 ) {
    echo "Tienes BIEN";
}
else {

    // Tercer if anidado
    if( $nota < 9 ) {
        echo "Tienes NOTABLE";
    }
    else {
        // Cuarto if anidado
        if( $nota <= 10 ) {
            echo "Tienes SOBRESALIENTE";
        }
        else {
            echo "Error en la nota";
            echo "Tiene que ser entre 0 y 10";
        }
    }
}
}
}
}

echo "Fin de la estructura if";
?>

```

En este fragmento de programa se puede apreciar la importancia del sangrado de código para la legibilidad de los programas.

Si hiciéramos la traza del programa, veríamos que emite el mensaje adecuado en función de la nota introducida. Sin embargo, el código es algo enrevesado y la mayoría de los programadores prefieren escribir la instrucción `if...else` anterior de la siguiente forma:

```

<?php
// Obtenemos la nota desde un formulario
$nota = $_POST['nota'];

if ( $nota >= 0 and $nota < 5 ) {
    // Se ha suspendido
    echo "¡¡Suspendido!!<br>";
    echo "¡¡A recuperar!!";
}
elseif( $nota < 6 ) {
    echo "Tienes SUFICIENTE";
}
elseif( $nota < 7 ) {
    echo "Tienes BIEN";
}
elseif( $nota < 9 ) {
    echo "Tienes NOTABLE";
}
elseif( $nota <= 10 ) {
    echo "Tienes SOBRESALIENTE";
}

```



```
}  
else {  
    echo "Error en la nota";  
    echo "Tiene que ser entre 0 y 10";  
}  
echo "Fin de la estructura if";  
?>
```

Puede haber varios `elseif` dentro de la misma sentencia `if`. La primera expresión `elseif` (si hay alguna) que se evalúe como `true` sería ejecutada. En PHP también se puede escribir `else if` (en dos palabras) y el comportamiento sería idéntico al de `elseif` (en una sola palabra). El significado sintáctico es ligeramente diferente (si se está familiarizado con C, este es el mismo comportamiento) pero la conclusión es que ambos resultarían tener exactamente el mismo comportamiento.

La sentencia `elseif` es ejecutada solamente si la expresión `if` precedente y cualquiera de las expresiones `elseif` precedentes son evaluadas como `false`, y la expresión `elseif` actual se evalúa como `true`.

#### 1.2.4 Operador ?:

Existe una versión abreviada de la estructura `if .. else` para aquellos casos en los que tengamos que generar un valor en función del resultado de una condición. Consiste en el operador ternario `?`. Es ternario por que necesita tres operandos. Su sintaxis es la siguiente:

```
<expresión_lógica> ? <expresión_verdadero> : <expresión_falso>;
```

Donde

- ✓ `<expresión_lógica>` → Expresión relacional o lógica a evaluar.
- ✓ `<expresión_verdadero>` → Expresión que se evalúa y devuelve su valor si la expresión lógica es `true`.
- ✓ `<expresión_falso>` → Expresión que se evalúa y devuelve su valor si la expresión lógica es `false`.

Si la expresión lógica que precede a `?` es verdadera, la expresión que se evalúa y cuyo valor arroja como resultado es la que precede a los `:`. Si, por el contrario, la expresión lógica que precede a `?` es falsa, la expresión a evaluar es la que sucede a `:`. Las expresiones para la versión verdadera y falsa de esta construcción pueden ser de cualquier tipo y no tienen por que ser del mismo tipo, aunque generalmente lo son.

Normalmente el valor generado resultado de evaluar una de las dos expresiones se asigna a una variable o se emplea como operando en una expresión más general. En el primer caso cuando se emplea este operador suele hacerse así

```
$<variable> = (<exp_lógica> ? <exp_verdadero> : <exp_falso>);`
```

es equivalente a:

```
if (<exp_lógica>)  
    $<variable> = <exp_verdadero>;  
else  
    $<variable> = <exp_falso>;
```

Veamos un ejemplo.

```
<?php  
$genero = $_POST['genero'];  
$altura_minima = $genero == "F" ? 165 : 175;  
  
echo "La altura mínima para el género $genero es $altura_minima";  
?>
```

### 1.2.5 Operador de fusión de null

El operador ?? en PHP es conocido como el operador de fusión de null (*null coalescing operator*). Se introdujo en PHP 7 y es muy útil para comprobar si una variable existe y tiene un valor distinto de null.

Este operador devuelve el primer valor que no sea null entre los operandos que se le proporcionen. En términos más simples, si la variable o expresión a la izquierda del ?? existe y no es null, se devuelve ese valor. Si no, se devuelve el valor de la derecha. Su sintaxis es:

```
$variable = $a ?? $b;
```

- ✓ Si \$a existe y no es null, entonces \$variable será igual a \$a.
- ✓ Si \$a no existe o es null, entonces \$variable será igual a \$b.

Por ejemplo:

```
$nombre = null;  
$nombreUsuario = $nombre ?? 'Invitado';  
echo $nombreUsuario; // Salida: 'Invitado'
```

Aquí la variable \$nombre es null. Si fuera al contrario

```
$nombre = "Juan";  
$nombreUsuario = $nombre ?? 'Invitado';  
echo $nombreUsuario; // Salida: 'Juan'
```

Puedes usar el operador ?? de manera encadenada para verificar múltiples valores:

```
$nombre = null;  
$apellido = null;  
$usuario = 'Usuario123';  
  
$nombreUsuario = $nombre ?? $apellido ?? $usuario ?? 'Invitado';  
echo $nombreUsuario; // Salida: 'Usuario123'
```

En este caso, tanto \$nombre como \$apellido son null, pero \$usuario tiene el valor 'Usuario123', así que ese valor es el que se asigna.

Aunque los operadores `?` y `??` parecen similares, tienen una diferencia importante:

- ✓ El operador ternario (`?:`) verifica si una expresión es verdadera o falsa. En PHP, no solo `null` es considerado como `false`, sino también valores como `0`, `false`, cadenas vacías, etc.
- ✓ El operador de fusión de `null` (`??`) solo verifica si una expresión es `null`. Cualquier otro valor (incluyendo `false`, `0`, o una cadena vacía) será considerado válido.

Por ejemplo:

```
$valor = 0;

$resultadoTernario = $valor ?: 'Predeterminado'; // Salida:
'Predeterminado'
$resultadoNullCoalescing = $valor ?? 'Predeterminado'; //
Salida: 0
```

Con el operador ternario (`?:`), el valor `0` es considerado falso, así que se devuelve 'Predeterminado'. Con el operador de fusión de `null` (`??`), `0` no es `null`, por lo que se devuelve `0`.

El operador `??` te permite asignar un valor de respaldo (*fallback*) si una variable es `null`. Es ideal para manejar variables no definidas o valores que pueden ser `null`, especialmente en formularios, bases de datos, o cuando recibes datos de usuarios.

### 1.2.6 Selectiva múltiple. Sentencia `switch`

En apartados anteriores hablamos sobre la instrucción `if` de selección simple y la instrucción `if...else` de selección doble. PHP cuenta con la instrucción `switch` de selección múltiple para realizar distintas acciones, según los posibles valores de una variable o expresión. Cada acción se asocia con un valor, indicado en una cláusula `case`, de una expresión entera constante, es decir, un valor constante de tipo `integer`. También se permite que la expresión sea de tipo cadena (`string`). El tipo de cada valor debe ser compatible con el tipo de la expresión en la que se basa la instrucción `switch`. No se permiten valores duplicados.

La sintaxis de esta instrucción es la siguiente:

```
switch ( <expresión> ) {

    case <constante_1>:
        <instrucción_1>;
        break;

    case <constante_2>:
        <instrucción_2>;
        break;

    ...

    case <constante_n>:
```

```
<instrucción_n>;  
break;  
  
[ default:  
  <instrucción_por_defecto>; ]  
}
```

Donde

- ✓ **<expresión>** → Expresión a comparar en cada cláusula `case`.
- ✓ **<constante\_1>** → Valor literal de tipo compatible a **<expresión>** que se compara con el valor de esta.
- ✓ **<instrucción\_1>** → Instrucción a ejecutar si el valor literal **<constante\_1>** coincide con el valor de **<expresión>**.
- ✓ **break** → Se rompe el flujo del programa para continuar con la siguiente instrucción posterior a la estructura `switch`.
- ✓ **<instrucción\_por\_defecto>** → Instrucción que se ejecuta cuando ningún valor de las cláusulas `case` coincide con el valor de la **<expresión>**.

Las instrucciones pueden ser simples o bloques de código.

El funcionamiento de esta estructura selectiva múltiple es el siguiente. Se evalúa la **<expresión>** la cual tendrá un valor. Se compara el valor generado con el literal de tipo compatible que va en la primera cláusula `case`. Si coinciden se ejecuta la **instrucción\_1** y se continua ejecutando el código por debajo salvo que se encuentre la sentencia `break` que provoca la terminación de la sentencia `match` y el flujo del programa continua por la siguiente sentencia al `match`.

Por tanto, cuando encuentra una coincidencia, se ejecuta el código asociado al bloque `case` y el código de todos los bloques `case` subsiguientes, aunque no se sigue comparando con el resto de literales de las restantes cláusulas `case`. Si queremos evitarlo tendremos que añadir la sentencia `break`.

Si el valor generado en la **<expresión>** no coincide con el literal del primer `case`, se compara con el literal de la segunda cláusula `case`. Si coincide se ejecuta su correspondiente código asociado y el código subsiguiente en el resto de cláusulas `case`, salvo que se rompa con un `break`. Así sucesivamente, hasta que encuentre un literal en alguna cláusula `case` que coincida con el valor generado.

Si no coincide el valor generado con ningún literal de ninguna cláusula `case` entonces no se ejecuta ninguna instrucción. Salvo que se haya incluido una cláusula opcional `default`, en cuyo caso se ejecutarían sus sentencias asociadas.

Veamos un ejemplo. En el siguiente programa se evalúa una nota numérica entera para traducirla a una calificación.

```
<?php
```

```
// Obtenemos la nota desde un formulario
$nota = $_POST['nota'];

switch ( $nota ) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
        echo "Suspenso";
        break;

    case 5:
        echo "Suficiente";
        break;

    case 6:
        echo "Bien";
        break;

    case 7:
    case 8:
        echo "Notable";
        break;

    case 9:
    case 10:
        echo "Sobresaliente";
        break;

    default:
        echo "La nota no es válida";
}
?>
```

Analicemos el código anterior. La expresión de la instrucción `switch` es únicamente una variable de tipo entero, `$nota`, por tanto simplemente devuelve su valor. Ahora cada valor literal de una cláusula `case` se compara con el valor de `nota`, comenzando por el primero. Si coincide ejecuta la sentencia que visualiza la calificación. En caso de no introducir un número entre 0 y 10, entonces se ejecuta la sentencia asociada a la cláusula `default`.

Nótese que debajo de la sentencia hay una instrucción `break`, la cual provoca que el flujo del programa continúe en la siguiente instrucción a la estructura `switch`. Si esta instrucción `break` no se pusiera, se ejecutarían todas las sentencias que hay debajo de la cláusula **case** con el valor coincidente, incluyendo aquellas cuyo valor de `case` no coincide con el de la expresión evaluada en `switch`. De ahí lo útil que es omitir el `break` en las cláusulas `case` con valor 0 al 3, ya que si coincidiera con alguno se ejecutaría la instrucción del `case` con valor 4 y justo después se encuentra el `break` que provocaría la salida del flujo del programa de la estructura `switch`.

### 1.2.7 Sentencia match

La sentencia `match` es parecida a `switch` pero con algunas diferencias. Veamos primero su sintaxis:

```
$<variable> = match ( <expresión> ) {  
    <exp_comp_1> => <expresión_retorno1>,  
    <exp_comp_2> => <expresión_retorno2>,  
    <exp_comp_3>,<exp_comp_4>, ... => <expresión_retorno3>,  
    ...  
    default => <expresión_retorno_defecto>  
};
```

Donde

- ✓ `<expresión>` → Expresión a comparar con cada valor constante.
- ✓ `<exp_comp_1>` → Expresión cuyo resultado se compara con `<expresión>`. La comparación es de igualdad estricta en tipo con el operador `===`.
- ✓ `<expresión_retorno_1>` → Valor que se devuelve si `<exp_comp_1>` coincide con el valor de `<expresión>`. Se pueden indicar tantos pares de expresiones de comparación y sus respectivas expresiones de retorno. Incluso podemos incluir más de una expresión de comparación, separadas por coma, con una única expresión de retorno
- ✓ `default => <expresión_retorno_defecto>` → Expresión que se devuelve si ningún valor de las expresiones de comparación coincide con el valor de la `<expresión>`.

Las características de `match` son:

- ✓ `match` es una expresión, por lo tanto genera un resultado que puede ser asignado a una variable o devuelto.
- ✓ Cada variante de `match` solo acepta una expresión de una sola línea y por lo tanto no necesita `break` como en el caso de `switch`.
- ✓ Las comparaciones de `match` son estrictas en tipo. Es decir, realiza igualdades con el operador `===`. Por tanto `'8.0'` (cadena) no coincidirá con `8.0` (número).
- ✓ Se puede poner más de un valor separador por comas.
- ✓ Una expresión `match` debe ser completa. Si la expresión no coincide con ningún valor de comparación se lanza un `UnhandledMatchError`.

Vamos a repetir el ejemplo anterior que vimos con `switch`. Se apreciará que queda mucho más abreviado.

```
<?php  
// Obtenemos la nota desde un formulario  
$nota = $_POST['nota'];
```

```
$calificación = match( $nota ){
    0, 1, 2, 3, 4  => "Suspenso",
                    5  => "Suficiente",
                    6  => "Bien",
                    7, 8  => "Notable",
                    9, 10 => "Sobresaliente",
    default => "Nota no válida",
};
echo "La nota ha sido $nota y la calificación es $calificación";
?>
```

## 1.3 Bucles

Un bucle es una estructura de programación que repite la ejecución de un conjunto de instrucciones varias veces, en función de una expresión relacional o lógica. Cada repetición de la ejecución del conjunto de instrucciones se denomina iteración.

En cada iteración hay que evaluar de nuevo la condición del bucle para comprobar si hay que seguir iterando. Dependiendo del bucle, esta comprobación se hace antes de ejecutar una nueva iteración, o después.

Obviamente, entre las instrucciones del bucle, hay que modificar los valores que intervienen en la condición del bucle para que en algún momento tenga un resultado `false`, ya que de lo contrario, se continuarían ejecutando iteraciones y el bucle no acabaría nunca. Un bucle infinito provoca el bloqueo de la aplicación.

Tenemos cuatro estructuras de control repetitivas. El uso de una u otra dependerá de las necesidades en cada caso, ya que en función de lo que se pretenda viene mejor utilizar una que otra.

### 1.3.1 For

Un bucle `for` ejecuta un conjunto de instrucciones un número predeterminado de veces que está controlado por una variable contador. Generalmente, la variable contador se inicializa con un valor, generalmente 0 o 1, y en cada iteración se incrementa otro valor, generalmente uno, además de comprobar si la variable contador ha alcanzado un valor que determina el fin del bucle.

La sintaxis de la estructura `for` es la siguiente

```
for( <inicio>; <comprobar_contador>; <incrementa_contador> )
    <instrucción>;
```

Donde

- ✓ `<inicio>` → Expresión para inicializar una variable de tipo entero que controla el número de iteraciones a ejecutar.
- ✓ `<comprobar_contador>` → Expresión relacional que comprueba si la variable contador ha alcanzado, o sobrepasado, el valor máximo que determina el fin del bucle.

- ✓ `<incrementa_contador>` → Expresión aritmética que aumenta el valor, generalmente en 1, de la variable contador.
- ✓ `<instrucción>` → Instrucción a ejecutar en el cuerpo del bucle. Puede ser una instrucción simple o un bloque de código.

Veamos un ejemplo y luego lo analizamos.

```
<?php
for ( $contador = 1; $contador <= 10; $contador++ ) {
    echo $contador;
}
?>
```

Con este bucle `for` sencillo se muestra por pantalla los 10 primeros números enteros. La expresión de inicio es `$contador = 1`. Se declara una variable llamada `$contador` y se inicializa a 1. Esta expresión solamente se ejecuta una vez, al inicio del bucle.

La expresión relacional `$contador <= 10` comprueba si la variable `$contador` ha alcanzado el valor 10. Si la expresión es `true`, se ejecuta una iteración más. Si es `false`, se termina el bucle y el flujo del programa continua en la siguiente instrucción al bucle `for`. Esta expresión se evalúa antes de ejecutar una iteración.

La expresión aritmética `$contador++` incrementa en uno el valor de la variable `$contador`. Se ejecuta cuando se termina la ejecución de la iteración actual y antes de volver a evaluar la expresión relacional que comprueba su valor. De esta forma, al modificar la variable contador sabemos que en algún momento la expresión relacional que controla la ejecución de una nueva iteración será `false` y terminará el bucle.

La variable `$contador` puede usarse en el cuerpo de la estructura `for` como otra variable cualquiera, pero se recomienda muy encarecidamente que no se cambie su valor, ya que de lo contrario podría alterarse de forma indeseada el número de iteraciones a ejecutar. Generalmente las variables contador se accede a su valor en lectura.

Por ejemplo, vamos a visualizar por pantalla la tabla de multiplicar de un número.

```
<?php
$numero = $_POST['numero'];

for ( $contador = 1; $contador <= 10; $contador++ ) {
    echo "$numero x $contador = {strval($numero*$contador)}";
}
?>
```

Si la expresión de inicialización en el encabezado del `for` declara la variable contador (es decir, si el tipo de la variable de control se especifica antes del nombre de la variable, como en el ejemplo), la variable de contador puede utilizarse sólo en esa instrucción `for`; no existirá fuera de esta instrucción.

Las tres expresiones en un encabezado `for` son opcionales. Si se omite `<comprobar_contador>`, PHP asume que esta condición siempre será verdadera, con lo cual se crea un ciclo infinito. Podríamos omitir la expresión de inicio si el programa inicializa



la variable contador antes del ciclo. Podríamos omitir la expresión de incremento si el programa calcula el incremento mediante instrucciones dentro del cuerpo del bucle, o si no se necesita un incremento. Veamos los siguientes ejemplos.

```
<?php

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

La expresión de incremento en un `for` actúa como si fuera una instrucción independiente al final del cuerpo del `for`. Por lo tanto, las expresiones

```
$contador = $contador + 1;
$contador += 1;
++$contador;
$contador++;
```

son expresiones de incremento equivalentes en una instrucción `for`. Muchos programadores prefieren `$contador++`, ya que es concisa y además un bucle `for` evalúa su expresión de incremento después de la ejecución de su cuerpo. Por ende, la forma de postincremento parece más natural. En este caso, la variable que se incrementa no aparece en una expresión más grande, por lo que los operadores de preincremento y postdecremento tienen en realidad el mismo efecto.

El incremento de una instrucción `for` también puede ser negativo, en cuyo caso sería un decremento y el bucle contaría en orden descendente.

Si al principio del bucle la condición de continuación de bucle es `false`, el programa no ejecutará el cuerpo de la instrucción `for`, sino que la ejecución continuará con la instrucción que siga inmediatamente después del `for`.

Con frecuencia, los programas envían a la salida el valor de la variable de control o lo utilizan en cálculos dentro del cuerpo del ciclo, pero este uso no es obligatorio. Por lo general, la variable de control se utiliza para controlar la repetición sin que se le mencione dentro del cuerpo del `for`.

El bucle `for` es muy flexible. Normalmente la variable contador se inicializa con un valor bajo, típicamente 0 o 1, y luego se incrementa en uno hasta alcanzar el número de iteraciones que se pretende. Sin embargo, también podemos inicializar con cualquier valor, incrementar o decrementar con una cantidad diferente a uno y la condición de salida puede ser cualquiera, inclusive una que no implique a la variable contador.

Los siguientes ejemplos muestran técnicas para modificar la variable de control en una instrucción `for`. En cada caso, escribimos el encabezado `for` apropiado. Observe el cambio en el operador relacional para los bucles que decrementan la variable de control.

```
<?php
// Modificar la variable de control de 1 a 100
// en incrementos de 1.
for ( $i = 1; $i <= 100; $i++ ) {
    echo $i;
}

// Modificar la variable de control de 100 a 1
// en decrementos de 1.
for ( $i = 100; $i >= 1; $i-- ) {
    echo $i;
}

// Modificar la variable de control de 7 a 77
// en incrementos de 7.
for ( $i = 7; $i <= 77; $i += 7 ) {
    echo $i;
}

// Modificar la variable de control de 20 a 2
// en decrementos de 2.
for ( $i = 20; $i >= 2; $i -= 2 ) {
    echo $i;
}

// Modificar la variable de control con
// la siguiente secuencia de valores: 2, 5, 8, 11, 14, 17, 20.
for ( $i = 2; $i <= 20; $i += 3 ) {
    echo $i;
}

// Modificar la variable de control con la siguiente secuencia
// de valores: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.
for ( $i = 99; $i >= 0; $i -= 11 ) {
    echo $i;
}
?>
```

Las expresiones de inicialización e incremento pueden ser listas separadas por comas de expresiones que nos permitan utilizar varias expresiones de inicialización, o varias expresiones de incremento. Por ejemplo

```
<?php
for ( $n = 2; $n <= 20; $total += $n, $n += 2 ) {
    echo "$n $total";
}
?>
```

Existe una variante del bucle `for` que se emplea para recorrer los elementos de un array y sobre objetos. Se verá posteriormente cuando veamos los arrays.

### 1.3.2 While

En una instrucción repetitiva `while` contiene una expresión booleana. Si es `true`, se ejecuta una iteración del bucle. Si es `false`, se sale del bucle y el flujo del programa continua por la siguiente instrucción inmediata al `while`. Por tanto, el número de iteraciones es indeterminado. La sintaxis es:

```
while( <condición> )
    <instrucción>;
```

Donde

- ✓ **<condición>** → Expresión booleana de permanencia en el bucle. Si es `true` se ejecutan las instrucciones del cuerpo del bucle. Si es `false`, el bucle se termina.
- ✓ **<instrucción>** → Instrucción a ejecutar en el cuerpo del bucle. Puede ser una instrucción simple o un bloque de código.

Si al ejecutar el bucle por primera vez la expresión booleana de la condición es `false`, no se ejecutaría el cuerpo del bucle y no se producirían iteraciones.

```
<?php
/* ejemplo 1 */

$i = 1;
while ($i <= 10) {
    echo $i++; /* el valor presentado sería
                $i antes del incremento
                (post-incremento) */
}
?>
```

Es evidente que en el cuerpo del bucle habrá que modificar los valores de las variables que intervengan en la condición del bucle, ya que en algún momento tendría que hacerse `false` para salir del bucle. De lo contrario tendríamos un bucle infinito.

### 1.3.3 Do.. While

La instrucción de repetición `do...while` es similar a la instrucción `while`. La diferencia estriba en que evalúa la condición de continuación de bucle después de ejecutar el cuerpo del bucle; por lo tanto, el cuerpo siempre se ejecuta por lo menos una vez.

Su sintaxis es:

```
do
    instrucción;
while( <condición> );
```

Donde

- ✓ **<condición>** → Expresión booleana de permanencia en el bucle. Si es `true` se ejecutan las instrucciones del cuerpo del bucle. Si es `false`, el bucle se termina.
- ✓ **<instrucción>** → Instrucción a ejecutar en el cuerpo del bucle. Puede ser una instrucción simple o un bloque de código.

Podemos modificar el ejemplo anterior, y quedaría de la siguiente manera con otro bucle `do .. while`.

```
<?php
$i = 0;
do {
    echo $i;
} while ($i > 0);
?>
```

El bucle de arriba se ejecutaría exactamente una sola vez, ya que después de la primera iteración, cuando la expresión verdadera es verificada, se evalúa como `false` (`$i` no es mayor que 0) y termina la ejecución del bucle.

### 1.3.4 Romper el flujo de control. Instrucciones `break` y `continue`

Además de las instrucciones de selección y repetición, PHP cuenta con las instrucciones `break` y `continue` para alterar el flujo de control. En epígrafes anteriores mostramos cómo puede utilizarse la instrucción `break` para terminar la ejecución de una instrucción `switch`. En esta sección veremos cómo utilizar `break` en las instrucciones de repetición.

Cuando `break` se ejecuta en una instrucción `while`, `for`, `do...while`, o `switch`, ocasiona la salida inmediata de esa instrucción. La ejecución continúa con la primera instrucción después de la instrucción de control. Los usos comunes de `break` son para salir anticipadamente del ciclo, o para omitir el resto de una instrucción `switch`. Por ejemplo

```
<?php

while( true ) {
    $numero = rand(-100,100);
    if( $numero < 0 )
```

```
    break;

    echo $numero;
}
?>
```

Cuando se genera un número negativo se sale del bucle mediante `break`, lo que provoca que no llegue a visualizarse en el cuerpo del bucle. Así, solamente mostrará los números positivos generados.

`break` acepta un argumento numérico opcional que indica de cuántas estructuras anidadas circundantes se debe salir. El valor predeterminado es 1, es decir, solamente se sale de la estructura circundante inmediata. Por ejemplo

```
<?php
$i = 0;
while ( ++$i ) {
    switch ( $i ) {
        case 5:
            echo "En 5<br />\n";
            break 1; /* Sólo sale del switch. */
        case 10:
            echo "En 10; saliendo<br />\n";
            break 2; /* Sale del switch y del while. */
        default:
            break;
    }
}
?>
```

Cuando la instrucción `continue` se ejecuta en una instrucción `while`, `for` o `do...while`, omite las instrucciones restantes en el cuerpo del ciclo y continúa con la siguiente iteración del ciclo. En las instrucciones `while` y `do...while`, la aplicación evalúa la prueba de continuación de ciclo justo después de que se ejecuta la instrucción `continue`. En una instrucción `for` se ejecuta la expresión de incremento y después el programa evalúa la prueba de continuación de ciclo.

Modificamos el ejemplo anterior para que omita la visualización del 0, ya que ni es positivo ni negativo.

```
<?php
while( true ) {
    $numero = rand(-100,100);

    if( $numero == 0 )
        continue;

    if( $numero < 0 )
        break;

    echo $numero;
}
```

```
}  
?>
```

Vemos que después de generar el número comprueba si es cero. Si es así, la instrucción `continue` salta a la comprobación de la condición del bucle sin ejecutar ninguna instrucción más. El resultado de la condición determinará si se continua en el bucle o no.

## 1.4 Sintaxis alternativas de las estructuras de control

PHP ofrece una sintaxis alternativa para algunas de sus estructuras de control: `if`, `while`, `for`, `foreach`, y `switch`. En cada caso, la forma básica de la sintaxis alternativa es cambiar la llave de apertura por dos puntos (`:`) y la llave de cierre por `endif;`, `endwhile;`, `endfor;`, `endforeach;`, o `endswitch;` respectivamente.

```
<?php if ($a == 5): ?>  
A es igual a 5  
<?php endif; ?>
```

En el ejemplo anterior, el bloque HTML "A es igual a 5" está anidado dentro de una sentencia `if` escrita en la sintaxis alternativa. El bloque HTML se mostraría solamente si `$a` es igual a 5.

La sintaxis alternativa también se aplica a `else` y `elseif`. La siguiente es una estructura `if` con `elseif` y `else` en el formato alternativo:

```
<?php  
if ($a == 5):  
    echo "a igual 5";  
    echo "...";  
elseif ($a == 6):  
    echo "a igual 6";  
    echo "!!!";  
else:  
    echo "a no es 5 ni 6";  
endif;  
?>
```

## 2 Arrays

Un array es un conjunto de variables que están relacionadas y se referencian con el mismo nombre. Cada variable en el array se conoce como elemento o componente del array. Cada componente del array tiene asociado una clave para acceder a un elemento específico en un array. En PHP un array está optimizado para varios usos diferentes; se puede emplear como un array tradicional, una lista, una tabla asociativa (tabla hash), un diccionario, una colección, una pila, una cola, etc. Ya que los valores de un array pueden ser otros arrays, también son posibles árboles y arrays multidimensionales.

Cuando queremos referir un elemento del array tenemos que usar la variable array y poner entre corchetes su clave.

```
$variable[<clave>]
```

La clave empleada puede ser un número entero. En este caso estamos ante un array escalar y la clave se conoce como índice del elemento, pudiendo ser una expresión de tipo entero. El primer elemento en el array es el elemento con índice 0.

Si la clave es una cadena de caracteres entonces estamos ante un array asociativo y el índice es una expresión de tipo cadena.

## 2.1 Arrays escalares

Se puede declarar un array con la función `array()`. Esta función tiene una lista de parámetros que serían los elementos del array. Por ejemplo

```
<?php
$notas = array(0 => 7.5, 1 => 8, 2 => 3, 3 => 2.5, 9);
?>
```

Como vemos en el ejemplo cada elemento del array se define con `<índice> => <valor>`. En el caso de los arrays escalares, el índice es una expresión de tipo entero y el valor es una expresión de cualquier tipo.

En la función `array()` pueden omitirse los índices en cuyo caso asigna automáticamente en orden un índice entero comenzando por cero. El ejemplo anterior equivale al siguiente:

```
<?php
$notas = array(7.5, 8, 3, 9);
?>
```

Una sintaxis abreviada disponible para declarar un array sin utilizar la función `array()` es mediante el operador `[]`. El array anterior se podría haber declarado de la siguiente manera:

```
<?php
$notas = [7.5, 8, 3, 9];
?>
```

Es posible indicar el índice de algunos elementos y excluir los demás. Por ejemplo:

```
<?php
$notas = array(7.5, 4 => 8, 3, 9);
?>
```

En este caso el elemento 0 es 7.5, el 4 es 8, el 5 es 3, el 6 es 9. En cuanto se indique un índice, los siguientes elementos sin índice son relativos a éste.

Cuando queremos acceder a un elemento individual del array usamos su índice.

```
<?php
// Array con cuatro elementos. Índices 0 a 3
$notas = array(7.5, 8, 3, 9);
```

```
// Se muestra el segundo elemento. Índice 1
echo "El elemento 2 es $notas[1]";
?>
```

Para modificar el valor de un elemento del array, hacemos una asignación normal a ese elemento. Por ejemplo

```
<?php
// Array con cuatro elementos. Índices 0 a 3
$notas = array(7.5, 8, 3, 9);

// Se modifica el tercer elemento. Índice 2
$notas[2] = 5.5;

// Se muestra el segundo elemento. Índice 1
echo "El elemento 2 es $notas[1]";
?>
```

Si ya existiera un elemento con ese índice, se cambiaría el valor de su contenido, en caso contrario creará un nuevo elemento del array y se le asignaría como valor el resultado de la expresión detrás del signo igual, de la misma forma que ocurre con una variable simple.

Si realizamos una asignación a un elemento del array sin indicar su índice estamos añadiendo un nuevo elemento al final del array.

```
<?php
// Array con cuatro elementos. Índices 0 a 3
$notas = array(7.5, 8, 3, 9);

// Se añade el quinto elemento. Índice 4
$notas[] = 6.5;

echo "El elemento 5 es $notas[4]";
?>
```

Si el array no existe, entonces lo crea con un elemento con índice 0.

Si lo que queremos hacer es eliminar un elemento del array podemos pasarlo como parámetro a la función `unset()`. Por ejemplo

```
<?php
// Array con cuatro elementos. Índices 0 a 3
$notas = array(7.5, 8, 3, 9);

// Se añade el quinto elemento. Índice 4
$notas[] = 6.5;

// Se modifica el segundo elemento
$notas[1] = 1.5;

// Se borra el tercer elemento
unset($notas[2]);
```



```
echo "El elemento 5 es $notas[4]";  
?>
```

## 2.2 Arrays asociativos

Los elementos de los arrays asociativos tiene una cadena de caracteres como índice. Al declararlos con la función `array()` hay que indicar las claves de cada elemento, si se omite, se les asigna automáticamente un índice entero.

```
<?php  
// Array asociativo con cuatro elementos.  
$coches= array(  
    "1234ABC" => "Ford Fiesta",  
    "2345BCD" => "BMW X1",  
    "3456CDE" => "Renault Clio",  
    "4567DEF" => "Fiat 500X" );  
  
// Mostramos el segundo elemento  
echo "El segundo coche es $coches['2345BCD']";  
  
// Se añade el quinto elemento  
$coches["5678EFG"] = "Peugeot 408";  
  
// Se modifica el segundo elemento  
$coches["2345BCD"] = "Seat Arona";  
  
echo "El elemento 5 es $coches['5678EFG']";  
?>
```

El índice puede ser cualquier expresión de tipo cadena de caracteres, habitualmente suele ser una variable `string`. Obviamente, el nombre de la variable nunca se pone entre comillas.

## 2.3 Arrays mixtos

PHP permite utilizar también arrays mixtos. En este caso algunos elementos tienen índice numérico y otros tienen cadena de caracteres como índice. También aquí podemos utilizar valores de variables como índices. Veamos un ejemplo:

```
<?php  
$notas['nombre'] = "Juan Pérez";  
$notas[0] = 7.5;  
$notas['calificacion'] = "Notable";  
  
echo "El alumno $notas['nombre'] ha obtenido una nota de  
$notas[0] con una calificación de $notas['calificacion'];  
?>
```

## 2.4 Arrays bidimensionales

En este caso cada elemento del array es otro array. Por tanto, para acceder a cada elemento individual del array necesitamos dos claves. Los arrays bidimensionales se conocen como matriz o tabla. Accedemos a cada uno de los elementos por un nombre seguido

de dos parejas de corchetes que contienen las claves.

Los claves pueden ser de tipo entero, o índices, equivalen entonces al número de fila y columna que la celda ocupa en la tabla, o puede ser asociativos lo que equivaldría en alguna medida a usar como índices los nombres de la fila y de la columna.

Para definir una matriz o tabla también usamos la función `array()`.

```
<?php
$notas = array(
=> array(0 => 7.5, 1 => 8, 2 => 3, 3 => 2.5, 4 => 9),
=> array(0 => 2.5, 1 => 5, 2 => 6.75, 3 => 8.5, 4 => 4),
=> array(0 => 4.5, 1 => 9, 2 => 5, 3 => 4, 4 => 4.5)
);
?>
```

En este ejemplo hemos indicado las claves de cada elemento. Se observa que cada elemento tiene como valor otro array. En este caso todos los elementos son arrays con el mismo número de elementos, pero no es obligatorio que sea así. El mismo ejemplo pero omitiendo los índices de cada elemento, asignados automáticamente en orden.

```
<?php
$notas = array(
    array(7.5, 8, 3, 2.5, 9),
    array(2.5, 5, 6.75, 8.5, 4),
    array(4.5, 9, 5, 5, 4, 4.5)
);
?>
```

Podemos añadir nuevos elementos tanto al final del array como al final de cada fila. Por ejemplo

```
<?php
// Al segundo elemento le añadimos otra nota. Ahora tiene 6
$notas[1][] = 6.5;
?>
```

PHP asigna automáticamente como primer índice el valor que sigue al último asignado y, si es el primero que se define, le pondrá como índice 0. En este caso como había 5 elementos previos, este será el sexto. Si añadimos otro elemento así

```
<?php
// Añadimos una cuarta fila con un solo elemento. Sería [3][0]
$notas[][] = 5.75;
?>
```

Al poner `[]` en el primer índice añade un nuevo elemento al array (cuarta fila) y con el segundo `[]` le añadimos un elemento al final de esta cuarta fila, que al acabar de ser creada no tiene ningún elemento y se le añade el primero.

En el caso de los arrays asociativos actuamos de la misma forma. Cada elemento tiene una clave y un array como valor, y dentro de este cada elemento también tiene una clave y un valor. Veamos un ejemplo

```

<?php
// Array asociativo bidimensional con cuatro elementos.
$coches= array(
    "1234ABC" => array( 'modelo' => "Ford Fiesta",
                        'precio' => 20000,
                        'motor' => "Diesel"),
    "2345BCD" => array( 'modelo' => "BMV X1",
                        'precio' => 40000,
                        'motor' => "Diesel"),
    "3456CDE" => array( 'modelo' => "Renault Clio",
                        'precio' => 15000,
                        'motor' => "Gasolina"),
    "4567DEF" => array( 'modelo' => "Fiat 500X",
                        'precio' => 18000,
                        'motor' => "Diesel")
);

// Mostramos el modelo del segundo elemento
echo "El segundo coche es $coches['2345BCD']['modelo']";

// Se añade el quinto elemento
$coches["5678EFG"]["modelo"] = "Peugeot 408";
$coches["5678EFG"]["precio"] = "45000";
$coches["5678EFG"]["motor"] = "Gasoil";

// Mostramos todos los elementos
foreach( $coches as $matricula => $coche ) {
    echo "Vehículo $matricula: <br>";
    foreach( $coche as $clave => $valor ) {
        echo "$clave -> $valor";
    }
}
?>

```

## 2.5 Arrays multidimensionales

PHP permite el uso de arrays con dimensión superior a dos. Para modificar la dimensión del array basta con ir añadiendo nuevos índices.

```
$a[x][y][z]=valor;
```

asignaría un valor al elemento de índices x, y y z de un array tridimensional y

```
$a[x][y][z][w]=valor;
```

haría lo mismo, ahora con un array de dimensión cuatro. Pueden tener cualquier tipo de índices: escalares, asociativos y, también, mixtos.

## 2.6 El bucle foreach

El bucle `foreach` es específico de los arrays y aplicable a ellos tanto si son escalares como si son de tipo asociativo. Este bucle recorre todo el array comenzando por el primer elemento y en cada iteración maneja un elemento. Tiene dos posibles opciones. En una de ellas lee únicamente los valores contenidos en cada elemento del array. En el otro caso lee

además las claves del array. Su sintaxis es:

```
foreach( <array> as [<clave> =>] <valor> ){  
    <instrucciones>  
}
```

Donde:

- ✓ <array> → Es la variable de tipo array.
- ✓ <clave> → Es la variable empleada para almacenar la clave de cada elemento en cada iteración. Es opcional, si se omite estamos accediendo solamente a los valores de los elementos del array.
- ✓ <valor> → Valor del elemento en cada iteración.

Las variables <clave> y <valor> no requieren estar definidas anteriormente.

Las instrucciones escritas entre el bloque de instrucciones permiten el tratamiento o visualización de los valores obtenidos. La variable <valor> no podrá ser utilizada para asignarle un nuevo valor. Hemos de tener en cuenta que su valor se rescribe en cada iteración del bucle y que al acabar este sólo contendrá el último de los valores leídos. Veamos un ejemplo

```
<?php  
// Array asociativo con cuatro elementos.  
$coches= array(  
    "1234ABC" => "Ford Fiesta",  
    "2345BCD" => "BMV X1",  
    "3456CDE" => "Renault Clio",  
    "4567DEF" => "Fiat 500X" );  
  
// Mostramos todos sus elementos  
foreach( $coches as $matricula => $modelo ) {  
    echo "El coche con matrícula $matricula es $modelo";  
}  
?>
```

Para un array escalar podemos omitir la clave y acceder solo a los valores

```
<?php  
// Array con cuatro elementos. Índices 0 a 3  
$notas = array(7.5, 8, 3, 9);  
  
// Se añade el quinto elemento. Índice 4  
$notas[] = 6.5;  
  
// Se modifica el segundo elemento  
$notas[1] = 1.5;  
  
// Se borra el tercer elemento  
unset($notas[2]);  
  
echo "Los elementos del array";
```

```
foreach( $notas as $nota ) {
    echo "La nota es $nota";
}
?>
```

Cuando se trata de arrays bidimensionales la lectura de los valores que contienen sus elementos requiere el uso de dos bucles anidados. Cuando queremos acceder a los elementos de un array de este tipo el primer bucle obtiene en cada iteración un elemento que consiste en su primera clave y un array como valor. Este último es recorrido por el segundo bucle, obteniendo su segunda clave y su correspondiente valor. La sintaxis sería algo así:

```
<?php
foreach($array_bidimensional as $clave1 => $elemento_array) {
    echo "Clave del array bidimensional: $clave1";
    foreach($elemento_array as $clave2 => $valor) {
        echo "Clave de cada array: $clave2";
        echo "Valor del elemento: $valor";
    }
}
?>
```

En un ejemplo anterior ya usamos un recorrido sobre un array bidimensional.

```
<?php
// Array asociativo bidimensional con cuatro elementos.
$coches= array(
    "1234ABC" => array( 'modelo' => "Ford Fiesta",
                        'precio' => 20000,
                        'motor' => "Diesel"),
    "2345BCD" => array( 'modelo' => "BMV X1",
                        'precio' => 40000,
                        'motor' => "Diesel"),
    "3456CDE" => array( 'modelo' => "Renault Clio",
                        'precio' => 15000,
                        'motor' => "Gasolina"),
    "4567DEF" => array( 'modelo' => "Fiat 500X",
                        'precio' => 18000,
                        'motor' => "Diesel")
);

// Mostramos todos los elementos
foreach( $coches as $matricula => $coche ) {
    echo "Vehículo $matricula: <br>";
    foreach( $coche as $clave => $valor ) {
        echo "$clave -> $valor";
    }
}
?>
```

En el caso de arrays con dimensiones superiores sería necesario proceder del mismo modo, y habría que utilizar tantos bucles `foreach` como índices contuviera el array.

## 3 Funciones

---

Con las estructuras de control vistas en el epígrafe anterior hemos visto que podemos romper el flujo secuencial de ejecución de instrucciones, en todos los casos en función del resultado de una expresión relacional o lógica. Hay otra forma de desviar este flujo de ejecución de las instrucciones y es mediante el empleo de una función.

Una función es un conjunto de instrucciones que se empaquetan en un bloque con un nombre. Estas instrucciones se ejecutarán dentro de un script cuando se necesite y el número de veces que se necesite. Para ejecutar una función solo hay que utilizar su nombre en una expresión del programa. La ejecución de una función se denomina llamada o invocación de la función. Cuando se invoca una función, el flujo de control del programa se desvía a la primera instrucción de la función, y cuando termina de ejecutarse el flujo de control se devuelve al punto del programa en el que se hizo su invocación.

Es frecuente que la función necesite unos datos para poder ejecutar sus instrucciones. Estos datos se denominan parámetros o argumentos y se le pasan en el momento de ejecutar la función. Cuando termine de ejecutarse, es posible que la función devuelva un valor como resultado de la misma al punto de invocación, el cual podrá utilizarse en expresiones.

De forma general, hay tres tipos de funciones:

- ✓ Funciones internas, integradas o predefinidas → Son funciones que todo lenguaje de programación tiene definidas y el programador solamente tiene que utilizarlas cuando lo necesite. En PHP existen multitud de funciones integradas las cuales están clasificadas. Para saber que tarea realizan y los parámetros que necesitan, consultar la documentación de cada una.
- ✓ Métodos → Son funciones que están definidas dentro de una clase de objeto. Estas funciones se conocen como métodos y se verán en detalle en el capítulo dedicado a la programación orientada a objeto.
- ✓ Funciones de usuario → Cuando se codifica un programa, el desarrollador también puede crear sus propias funciones. Además, si el desarrollador crea una función que puede utilizar en varios programas, la puede introducir en un módulo e importarla cuando quiera utilizarla en otros programas. De esta forma se implementa el paradigma de la programación modular, el cual extiende lo visto hasta ahora que implementaba la programación estructurada.

En este epígrafe nos centraremos en las funciones de usuario e introduciremos el uso de las funciones integradas.

### 3.1 Definición de una función

Para poder utilizar una función es necesario definirla previamente, que consiste en establecer el nombre de la función, sus parámetros, si los tuviera, y escribir las instrucciones que la componen.

En PHP una función pueden estar definida dentro de cualquier script y en cualquier parte del documento, sin que tenga importancia alguna el lugar en el que se incluya la llamada a la misma.

También es posible, y bastante habitual, incluir funciones de uso frecuente en documentos externos de modo que pueden ser compartidas. En este caso, además de invocarla es necesario indicar a PHP el lugar donde debe buscarla. Esto se verá en el siguiente epígrafe.

Una función está formada por dos partes:

- ✓ **Cabecera** → Se indica el nombre de la función y la lista de parámetros o argumentos de entrada de la función. Cada parámetro se especifica con su nombre. Pueden especificarse tantos parámetros como se necesiten.
- ✓ **Cuerpo de la función** → Conjunto de sentencias que forma la función. Opcionalmente, la última es una instrucción que devuelve un valor al punto de llamada.

La sintaxis para definir una función es la siguiente:

```
function <nombre>([<par1>, <par2>, ... ]){  
    <instrucciones>  
    return [<expresion>;  
}
```

La cabecera de la función comienza con la palabra reservada `function` que se usa para definir funciones. Debe seguirle el nombre de la función y la lista de parámetros formales entre paréntesis. Estos parámetros son datos que la función necesita para realizar su labor. En la definición de la función se tendrá en cuenta lo siguiente:

- ✓ El nombre de la función, que debe seguir criterios similares a los de los nombres de variables, aunque en este caso no se antepone el símbolo `$` ni ningún otro.
- ✓ Opcionalmente, la función puede llevar un número indeterminado de parámetros. Estos van incluidos dentro de los paréntesis en una lista de identificadores separados por coma.
- ✓ Los paréntesis son obligatorios aunque no haya ningún parámetro definido.
- ✓ Las llaves de apertura `{` y cierre `}` delimitan las instrucciones que componen la función.
- ✓ Cualquier código PHP válido puede aparecer dentro de una función, incluso otras funciones y definiciones de clases, aunque no se recomienda esto último.

Por ejemplo, veamos la siguiente definición de función:

```
<?php  
function area( $base, $altura) {  
    $a = $base * $altura;  
}
```

```
return $a;
}
?>
```

En el siguiente ejemplo, se define la función después de su invocación:

```
<?php
$b = 4;
$a = 3;

$area = area($b, $a);

echo "El área es $area";

/*
Hemos escrito un script con una llamada
a la función area() que aún no está definida.
Tendremos que hacerlo, pero no importa
la parte del documento en la que lo hagamos.
*/

function area( $base, $altura) {
    $a = $base * $altura;
    return $a;
}
?>
```

## 3.2 Ejecución de la función

Las funciones PHP no se ejecutan hasta que sean invocadas. Para invocar una función la sintaxis es la siguiente:

```
<nombre>([<par1>, <par2>, ...]);
```

Al ser llamada con esta sintaxis, desde cualquier script, se ejecutarán las instrucciones contenidas en ella. La lista de argumentos corresponde a los parámetros de la función. Cada argumento se separa del siguiente por una coma. Cuando termine su ejecución el script PHP continuará ejecutando las sentencias siguientes a la de invocación de la función. Si la función devolviera un valor podrá ser almacenado en una variable para un posterior uso. En este caso la llamada a la función tendría la siguiente sintaxis:

```
$<variable> = <nombre>([<par1>, <par2>, ...]);
```

Aunque también es posible utilizar el valor que devuelve en una expresión directamente sin almacenarlo en una variable.

## 3.3 Parámetros y argumentos

Ya hemos visto anteriormente que una función puede necesitar datos para ejecutar sus instrucciones. Estos datos son conocidos como parámetros o argumentos. Un parámetro o argumento es un dato que necesita la función para la ejecución de sus sentencias. Al definir la función debemos indicar los parámetros que tendrá separados por comas. Esta definición de parámetros puede hacerse de varias formas, tal y como veremos más adelante.



Cuando se invoca una función hay que especificar los parámetros de la llamada en cuestión. Así, cada vez que se invoca la función se pueden pasar diferentes parámetros y la función se ejecutaría cada vez con diferentes datos.

Los términos parámetro y argumento se emplean en muchas ocasiones indistintamente para referirnos a lo mismo, pero no son lo mismo. En realidad hacen referencia a cada dato que utiliza la función, pero en momentos diferentes. Los parámetros de una función pueden ser:

- ✓ Formales → Los que se declaran en la cabecera de la función. Sólo pueden ser nombres de variables y son, a efectos de ámbito, variables locales a la función. Generalmente conocidos como **parámetros**.
- ✓ Reales → Los que se pasan a la función en el momento de la llamada. Generalmente conocidos como **argumentos**. Pueden ser cualquier expresión con el mismo tipo de datos, o tipo compatible, del parámetro formal correspondiente. La expresión más simple sería una variable, constante o literal. Sin embargo no hay límite en cuanto a la dificultad de la expresión a utilizar como parámetro real. Puede ser una combinación de operandos (variables y/o constantes) y operadores o incluso el valor devuelto por otra función.

Por tanto, los parámetros son los que se definen en la cabecera de la función y se utilizan dentro de ella. Los argumentos son los utilizados en la invocación de la función y cuyo valor se copia en el parámetro correspondiente. Esta vinculación entre el parámetro y el argumento no obliga en ningún caso que deban tener el mismo nombre.

Cada vez que se realiza la llamada a una función se establece una correspondencia entre los parámetros reales y los formales, mediante la cual el valor de los parámetros reales se copia en los parámetros formales en el mismo orden en el que aparecen. De ahí que se conozcan como parámetros o argumentos posicionales. Así se comunican los datos desde el script que invoca a la función.

### 3.3.1 Paso de parámetros por referencia

Por defecto, los argumentos de las funciones son pasados por valor (así, si el valor del argumento dentro de la función cambia, este no cambia fuera de la función). Para permitir a una función modificar sus argumentos, éstos deben pasarse por referencia.

Para hacer que un argumento a una función sea siempre pasado por referencia hay que anteponer al nombre del argumento el signo & en la definición de la función. Por ejemplo

```
<?php
function añadir_algo(&$cadena)
{
    $cadena .= 'y algo más.';
}

$cad = 'Esto es una cadena, ';
añadir_algo($cad);
```

```
echo $cad;      // imprime 'Esto es una cadena, y algo más.'  
?>
```

El `&` puede anteponerse tanto en la definición de la función como en la llamada a la función. La desventaja de ponerlo en la definición de la función es que el argumento siempre se pasará por referencia. Sin embargo, habrá situaciones en que convenga pasarlo por referencia y otras veces por valor.

La segunda de las opciones nos concede mayor libertad dado que permite usar una sola función y decidir en cada llamada la forma de pasar los parámetros.

### 3.3.2 Valores parámetros por defecto

Una función puede definir valores predeterminados para argumentos escalares simplemente añadiendo una asignación de valor en la definición del parámetro. Esto permite invocar la función sin especificar el correspondiente argumento. En este caso, el valor por defecto definido en la cabecera de la función sería el valor del parámetro. Por ejemplo:

```
<?php  
function hacer_café($tipo = "capuchino")  
{  
    return "Hacer una taza de $tipo.\n";  
}  
echo hacer_café();  
echo hacer_café(null);  
echo hacer_café("espresso");  
?>
```

El resultado del ejemplo sería:

```
Hacer una taza de capuchino.  
Hacer una taza de .  
Hacer una taza de espresso.
```

Debemos tener en cuenta lo siguiente:

- ✓ PHP también permite el uso de arrays y del tipo especial `null` como valores predeterminados.
- ✓ El valor predeterminado debe ser una expresión constante, no puede ser una variable, un miembro de una clase o una llamada a una función.
- ✓ Podemos combinar parámetros con valor por defecto y sin valor por defecto. En este caso los parámetros sin valor por defecto deben estar en la cabecera de la función antes que los parámetros con valor por defecto, para evitar ambigüedad en la invocación de la función.

### 3.3.3 Parámetros con tipo de datos

Las declaraciones de tipo permiten a las funciones requerir que los parámetros sean de cierto tipo durante una llamada. Si el valor dado es de un tipo incorrecto, se generará una excepción `TypeError`.

Para especificar una declaración de tipo, debe anteponerse el nombre del tipo al nombre del parámetro. Se puede hacer que una declaración acepte valores `null` si el valor predeterminado del parámetro se establece a `null`.

Los tipos válidos que se pueden utilizar son:

Tipo	Descripción
nombre de clase/interfaz	El parámetro debe ser una <code>instanceof</code> del nombre de la clase o interfaz dada.
<code>self</code>	El parámetro debe ser una <code>instanceof</code> de la misma clase donde está definido el método. Esto solamente se puede utilizar en clases y métodos de instancia.
<code>array</code>	El parámetro debe ser un <code>array</code> .
<code>callable</code>	El parámetro debe ser un <code>callable</code> válido.
<code>bool</code>	El parámetro debe ser un valor de tipo <code>boolean</code> .
<code>float</code>	El parámetro debe ser un número de tipo <code>float</code> .
<code>int</code>	El parámetro debe ser un valor de tipo <code>integer</code> .
<code>string</code>	El parámetro debe ser un <code>string</code> .
<code>iterable</code>	El parámetro debe ser un <code>array</code> o una <code>instanceof Traversable</code> .
<code>object</code>	El parámetro debe ser un <code>object</code> .

Por ejemplo

```
<?php
function area( float $base, float $altura) {
    $a = $base * $altura;
    return $a;
}

float $b = 3;
float $a = 4;
$area = area($b, $a);
echo "El área es $area";
?>
```

### 3.3.4 Lista de parámetros de longitud variable

¿Qué es lo que ocurre cuando necesitamos una función que reciba un número variable de argumentos? Por ejemplo, queremos definir una función que calcule la media aritmética de un conjunto de números, pero sin indicar a priori cuantos números serán. En este caso es mejor indicar en la cabecera de la función que va a recibir una lista de parámetros.

PHP tiene soporte para listas de argumentos de longitud variable en funciones definidas por el usuario. Esto se implementa utilizando el token `...`. Por ejemplo:

```
<?php
function media(...$números) {
    $suma = 0;
    foreach ($números as $n) {
```

```
        $suma += $n;
    }
    return $suma / func_num_args();
}

echo media(1, 2, 3, 4);
?>
```

Como se puede observar, la función integrada `func_num_args()` devuelve cuantos argumentos se han pasado en la invocación.

Se puede especificar argumentos posicionales normales antes del token `...`. En este caso, solamente los argumentos al final que no coincidan con un argumento posicional serán añadidos al array generado por `...`

También es posible añadir una declaración de tipo antes del símbolo `...`. Si está presente, todos los argumentos capturados por `...` deben ser objetos de la clase implicada.

### 3.3.5 Argumentos con nombre

Los argumentos con nombre permiten pasar argumentos en la invocación de la función utilizando el nombre del parámetro en lugar de la posición. Esta forma de paso de parámetros es más autoexplicativa ya que en la invocación de la función se detalla explícitamente qué parámetro recibe qué valor y además permite saltarnos valores por defecto arbitrariamente.

Los argumentos con nombre se pasan indicando en la invocación de la función el nombre del parámetro, el operador `:` y el valor del argumento. Veamos un ejemplo.

```
<?php
function area( float $base, float $altura) {
    $a = $base * $altura;
    return $a;
}

float $b = 3;
float $a = 4;
$area = area(altura : $a, base: $b);
echo "El área es $area";
?>
```

Nótese la siguiente línea

```
$area = area(altura : $a, base: $b);
```

Aquí invocamos la función anteponiendo a cada valor del argumento el nombre del parámetro sin `$`. Además, los argumentos tienen intercambiado el orden con respecto a la definición de la función, cosa que podemos hacer perfectamente ya que ahora los argumentos se identifican con su nombre de parámetro en lugar de por su posición.

Si hacemos la invocación de la función así ya no estamos restringidos a definir primero los parámetros sin valor por defecto y después los parámetros con valor por defecto, ya que

al incluir el nombre del parámetro en la invocación estamos identificando sin ambigüedad cuál es el parámetro al que pasamos el argumento.

### 3.4 Devolución de valores

Las funciones PHP pueden, y suelen, devolver un valor al punto de invocación. Para ello usamos la sentencia opcional `return` seguida de la expresión cuyo resultado queremos que sea devuelto al punto de invocación. Se puede devolver cualquier tipo, incluidos arrays y objetos. Esto causa que la función finalice su ejecución inmediatamente y pase el control de nuevo al punto de invocación. Por ejemplo

```
<?php
function area( $base, $altura) {
    $a = $base * $altura;
    return $a;
}

$b = 3;
$a = 4;

echo "El área es $area " . area($b, $a);
?>
```

Tal como podemos ver en el ejemplo, el valor devuelto por `return` pueden ser utilizado directamente en una expresión o asignado a una variable.

La sentencia `return` puede aparecer en la función en cualquier lugar y más de una vez, aunque por lo general aparece una como la última sentencia de la función. Cuando no es así, generalmente está dentro de una estructura condicional. Si no fuera así, todas las sentencias por debajo de `return` no se ejecutarían nunca.

En el caso de que necesitemos devolver más de un valor al punto de invocación necesitaremos incluirlos en un array y devolver este. Cuando lo vayamos a recoger en el punto de invocación podemos usar una variable array o asignar todos los valores a variables individuales con la función `list()`. En esta segunda opción habría que hacer algo así:

```
list(v1, v2,..) = funcion()
```

Las variables `v1`, `v2`, etc. recogerán los valores de los elementos del array devuelto por la función. En el siguiente ejemplo usamos la primera forma:

```
<?php
define("PI", 3.1415);

function circulo( $radio) {

    $resultado[] = 2 * PI * $radio;
    $resultado[] = PI * $radio ** 2;
    return $resultado;
}

$r = 5;
```

```
$area_y_circulo = circulo($r);

echo "La longitud de la circunferencia de radio $r es
      $area_y_circulo[0] ";
echo "El área del círculo con radio $r es $area_y_circulo[1] ";

?>
```

En este otro ejemplo usamos una lista

```
<?php
define("PI", 3.1415);

function circulo( $radio) {

    $resultado[] = 2 * PI * $radio;
    $resultado[] = PI * $radio ** 2;
    return $resultado;
}

$r = 5;
list($circunferencia, $area) = circulo($r);

echo "La longitud de la circunferencia de radio $r es
      $circunferencia ";
echo "El área del círculo con radio $r es $area";

?>
```

De forma similar a las declaraciones de tipo de argumento, las declaraciones de tipo de devolución especifican el tipo del valor que serán devuelto desde una función. Están disponibles los mismos tipos para las declaraciones de tipo de devolución que para las declaraciones de tipo de parámetro. El tipo de devolución se indica después de la cabecera de la función con : de prefijo. Por ejemplo

```
<?php
function area( $base, $altura): float {
    $a = $base * $altura;
    return $a;
}

$b = 3;
$a = 4;

echo "El área es $area " . area($b, $a);

?>
```

Los valores de retorno pueden ser marcados como anulables anteponiendo al tipo de devolución con un signo de cierre de interrogación. Esto significa que la función puede devolver un valor del tipo especificado o null. Por ejemplo

```
<?php
function area( $base, $altura): ?float {
    if( $base < 0 or $altura < 0 ) {
        echo "No se permiten valores negativos en la base o altura";
    }
}
```

```
    return null;
}

$a = $base * $altura;
return $a;
}

$b = 3;
$a = 4;

echo "El área es $area " . area($b, $a);
?>
```

## 3.5 Ámbito de las variables

Cuando ejecutamos una función debemos saber a qué datos podemos acceder y a cuáles no. El ámbito de una variable es el contexto dentro del que la variable está definida. La mayor parte de las variables PHP sólo tienen un ámbito simple. Este ámbito simple también abarca los ficheros incluidos y los requeridos. Por ejemplo:

```
<?php
$a = 1;
include 'b.inc';
?>
```

Aquí, la variable `$a` estará disponible dentro del script incluido `b.inc`. Sin embargo, el interior de las funciones definidas por el usuario crea un ámbito local a la función. Cualquier variable usada dentro de una función está, por omisión, limitada al ámbito local de la función. Por ejemplo:

```
<?php
$a = 1; /* ámbito global */

function test ()
{
    echo $a; /* referencia a una variable del ámbito local */
}

test();
?>
```

Este script no producirá salida, ya que la sentencia `echo` utiliza una versión local de la variable `$a`, a la que no se ha asignado ningún valor en su ámbito. En PHP, las variables globales deben ser declaradas globales dentro de la función si van a ser utilizadas dentro de dicha función.

### 3.5.1 La palabra clave global

Veamos el siguiente ejemplo

```
<?php
$a = 1;
$b = 2;
```

```
function Suma ()
{
    global $a, $b;

    $b = $a + $b;
}

Suma ();
echo $b;
?>
```

El script anterior producirá la salida 3. Al declarar `$a` y `$b` globales dentro de la función, todas las referencias a estas variables se referirán a la versión global. No hay límite al número de variables globales que se pueden manipular dentro de una función.

Un segundo método para acceder a las variables desde un ámbito global es usando el array `$GLOBALS`. El ejemplo anterior se puede reescribir así:

```
<?php
$a = 1;
$b = 2;

function Suma ()
{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Suma ();
echo $b;
?>
```

El array `$GLOBALS` es un array asociativo con el nombre de la variable global como clave y los contenidos de dicha variable como el valor del elemento del array. `$GLOBALS` existe en cualquier ámbito, esto ocurre ya que `$GLOBALS` es una superglobal.

### 3.5.2 Variables static

Otra característica importante del ámbito de las variables es la variable estática. Una variable estática existe sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito. Consideremos el siguiente ejemplo:

```
<?php
function test ()
{
    $a = 0;
    echo $a;
    $a++;
}

?>
```

Esta función tiene poca utilidad ya que cada vez que es llamada asigna a `$a` el valor 0 e imprime un 0. La sentencia `$a++`, que incrementa la variable, no sirve para nada, ya que



en cuanto la función finaliza, la variable `$a` desaparece. Para hacer una función útil para contar, que no pierda la pista del valor actual del conteo, la variable `$a` debe declararse como estática:

```
<?php
function test ()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

Ahora, `$a` se inicializa únicamente en la primera llamada a la función, y cada vez que la función `test ()` es llamada, imprimirá el valor de `$a` y lo incrementa.

### 3.5.3 Resumen del ámbito de variables

Resumamos lo ya comentado cuando tratamos el tema de las variables.

- ✓ Las funciones no leen valores de variables definidas fuera de su ámbito salvo que dentro de la propia función se definan de forma expresa como globales.
- ✓ Si una función modifica el valor de una variable global, el nuevo valor persiste después de ejecutar la función.
- ✓ Si dentro de una función se utiliza un nombre de variable idéntico al de otra externa a ella (sin definirla global) la nueva variable se inicia con valor nulo y los eventuales valores que pudiera ir conteniendo se pierden en el momento en que se acaba su ejecución.
- ✓ Una variable definida dentro de una función con la palabra clave `static` provoca que guarde su valor entre invocaciones de la función.

Dentro de la función se pueden definir tantas variables locales como se necesiten, teniendo en cuenta que se destruirá cuando acabe la ejecución de la función. Los parámetros de la función se comportan como variables locales a la función y también desaparecen cuando termina la ejecución de la función. Veamos un ejemplo:

```
<?php
# definamos dos variables y asignémosles un valor
$a=5;
$b=47;

# escribamos una funcion a1 y pidámosle que imprima sus valores
function a1 () {
    echo "Este es el valor de \$a en la función a1: $a <br>";
    echo "Este es el valor de \$b en la función a1: $b <br>";
}

# hagamos una llamada a la función anterior
# no nos escribirá ningún valor porque esas variables
# no pertenecen al ámbito de la función y serán
```

```

# consideradas como vacías en el ámbito de la función
a1();

# escribamos una nueva función, definamos como global $a
# y comprobemos que ahora si la hemos incluido en el ámbito
# de la función
function a2() {
    global $a;
    echo "Este es el valor de \$a en la función a2: $a <br>";
    echo "Este es el valor de \$b en la función a2: $b <br>";
}

# invoquemos esta nueva función y veamos que ahora
# si se visualiza el valor de $a pero no el de $b
a2();

# creemos una nueva función y ahora modifiquemos dentro de ella
# ambas variables
function a3() {
    global $a;
    $a += 45;
    $b -= 348;
    echo "Este es el valor de \$a en la función a1: $a <br>";
    echo "Este es el valor de \$b en la función a1: $b <br>";
}
# invoquemos la funcion a3
a3();

# comprobemos -desde fuera del ámbito de la función
# que ocurrió con los valores de las variables
echo "El valor de \$a HA CAMBIADO después de ejecutar a3 es: $a
<br>";
echo "El valor de \$b NO HA CAMBIADO después de ejecutar a3 es:
$b <br>";

# probemos que ocurre con una variable superglobal
# veremos que sin ser definida expresamente en a4
# si pertenece a su ámbito y por lo tanto
# visualizamos su contenido
function a4() {
    print "La superglobales si están: ".$_SERVER['SERVER_NAME'];
}

# invoquemos esta nueva funcion
a4();
?>

```

### 3.6 Definición de la función con código HTML

Existe otra opción de definición de funciones de usuario que puede resultar de mucho interés. En este caso la función se define en tres bloques:

- ✓ Definición de la función, llave de apertura y cierre del script PHP.
- ✓ Contenido de la función formado exclusivamente por código HTML, que se enviaría a

la salida directamente tal cual cuando fuera invocada la función que lo contiene.

- ✓ Cierre de la función (llave de cierre) contenido en un bloque PHP, es decir, entre las etiquetas de apertura y cierre de PHP. Cuando es invocada una función definida de esta forma, puedes verlo en el ejemplo, PHP se limita a escribir en el documento final los textos contenidos entre la etiqueta de apertura y cierre de la función.

Las funciones de esta forma son particularmente útiles para la construcción de espacios web que contienen una serie de páginas en las que se repiten las mismas estructuras, como las del siguiente ejemplo.

```
<?php
function Encabezado() { ?>

<!-- Hemos abierto un bloque PHP exclusivamente
para la cabecera de la función.
Todo lo que viene a continuación es código HTML -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Titulo de mi página</title>
    <meta name="viewport"
      content="width=device-width, initial-scale=1.0">
    <link type="text/css" rel="stylesheet" href="formato.css">
  </head>
  <body>
    <!-- Este nuevo bloque PHP es para insertar la llave
    cierre de la función. -->
  <?php
  }

  function Pie() { ?>
    </body>
  </html>
<?php
}
?>
```

Las dos funciones anteriores se pueden usar para crear una salida HTML que se envía al cliente.

```
<?php
Encabezado();
?>
<p>
Este es texto que aparecerá en el cuerpo de la página.
Está fuera de los scripts de php y será considerado
como un texto HTML.
</p>
<?php
Pie();
?>
```

## 3.7 Recursividad

PHP soporta la recursión, que consiste en el proceso de definir algo en términos de si mismo. En programación, la recursión es una cualidad que permite a un método invocarse a si mismo. Un método que se llama a si mismo se dice que es recursivo. El ejemplo clásico de la recursión es el cálculo del factorial de un número. El factorial de un número  $N$  es el producto de todos los enteros entre 1 y  $N$ . Por ejemplo, el factorial de 3 es  $1 \times 2 \times 3 = 6$ .

Si nos fijamos bien, podemos deducir que el factorial de  $N$  es igual a  $N$  por el factorial de  $N - 1$ . Según esto tendríamos la siguiente clase para calcular el factorial de un número.

```
<?php
function factorial( $n ) : int {
    if ( $n == 1 ) {
        $resultado = 1;
    }
    else {
        $resultado = $n * factorial($n - 1);
    }
    return resultado;
}

echo "El factorial de 3 es {factorial(3)}";
echo "El factorial de 5 es {factorial(5)}";
echo "El factorial de 7 es {factorial(7)}";
?>
```

La salida del programa anterior sería

```
El factorial de 3 es 6
El factorial de 5 es 120
El factorial de 7 es 5040
```

Si nos centramos en el código del método `factorial()` vemos que recibe como parámetro el número al que tenemos que calcular el factorial y lo que hace es multiplicar el número por el factorial de ese número menos uno, el cuál obtiene invocando de nuevo el método `factorial()` y pasándole como argumento `$n-1`.

Las sucesivas invocaciones de un método a si mismo tienen que terminar en algún momento y será cuando el número sea 1.

Las invocaciones sucesivas de un método recursivo siempre se encuentran en una estructura de control para evitar la recursividad infinita. En algún momento la condición de control se hace false lo que pararía las invocaciones recursivas.

## 3.8 Funciones internas

En cualquier lenguaje de programación se conoce como función interna, integrada o predefinida, a aquella que no es necesario definir ya que es propia del lenguaje y forma parte de su librería estándar. Solamente hay que invocarla cuando se necesite.

PHP dispone de un número muy elevado de funciones internas que pueden ser

incluidas dentro de los scripts. Las funciones se suelen agrupar en extensiones en función de la tarea que realizan.

La definición de las funciones se encuentra dentro de lo que se denomina extensión, que es un componente que almacena el código de estas funciones y tenemos que instalarlo antes de poder utilizar sus funciones. Debido a ello, para el uso de unas funciones específicas es necesario que en el servidor Web esté instalado la extensión correspondiente. Por ejemplo, si necesitamos utilizar las funciones para interactuar con una base de datos MySQL, tendremos que instalar la extensión `php-mysql`, el cual contiene las funciones para hacerlo. Si queremos generar documentos pdf tendremos que instalar su extensión, etc.

Para ver la definición de estas funciones resulta ideal la web que contiene la documentación PHP en la cual aparece su sintaxis junto con la explicación de la misma y algunos ejemplos. En <https://www.php.net/manual/es/funcref.php> disponemos de una referencia de las funciones agrupadas por extensiones.

## 4 Ficheros externos

Cuando creamos un script PHP podemos reutilizar código escrito y probado anteriormente. Para ello solamente tenemos que incluir su contenido en el script que estemos desarrollando.

De hecho, a la hora incluir contenido de un archivo externo a nuestro script no estamos limitados a código PHP en archivo con extensión `.php`, sino en realidad a cualquier archivo de texto plano, aunque por lo general suele ser código PHP. Esta característica nos permite modularizar nuestra aplicación. Cuando desarrollamos una aplicación es habitual que el código se distribuya en varios archivos con funciones, clases, definiciones, etc. Cuando queremos utilizarlos, se incluyen en scripts donde se necesite.

PHP dispone de funciones que permiten insertar en un documento una parte o la totalidad de los contenidos de otro. Esta opción resulta muy interesante, tanto desde el punto de vista operativo como en lo relativo a la seguridad. Para incluir el contenido de un archivo en nuestros scripts usamos las funciones `include()`, `require()`, `include_once()` y `require_once()`.

### 4.1 La función `include()` y `require()`

La sentencia `include()` incluye y evalúa el archivo especificado como parámetro. Su sintaxis es:

```
include(<path_archivo>);
```

Donde:

- ✓ `<path_archivo>` → Es la ruta, absoluta o relativa, del archivo a incluir. Si no se indica path, se buscará el archivo en los directorios indicados en la directiva `include_path` de `php.ini`. Si el archivo no se encuentra, finalmente verificará en el propio directorio del script que hace la invocación y en el directorio de trabajo

actual, antes de fallar. En este caso, emitirá una advertencia si no puede encontrar un archivo, pero continua la ejecución del script.

El parámetro `<path_archivo>` es una cadena que contiene el path y el nombre del fichero cuyos contenidos pretendemos incluir. Pueden incluirse ficheros con cualquier extensión aunque es muy habitual utilizar archivos con extensión `.inc.php`. La primera parte (`inc`) nos permitirá identificar este tipo de ficheros mientras que la extensión `php` obligaría a que si un usuario malicioso pretende visualizar el contenido del fichero fuera interpretado por PHP y, como consecuencia de ello, solo devolvería el resultado sin permitir la visualización del código fuente.

Este tipo de ficheros pueden contener: texto, etiquetas HTML y funciones. Si no contiene funciones se podrá insertar tantas veces como se invoque y se insertará, además, todo su contenido tal como puedes ver en el ejemplo.

Si el fichero contiene funciones, clases o definiciones, solo podrá ser invocado una vez ya que si se hiciera una segunda llamada se produciría un error por duplicidad en los nombres de las funciones.

Generalmente, las invocaciones de `include()` se hacen al principio del script, en las primeras líneas, para poder disponer de todas sus definiciones a partir de entonces.

Cuando un archivo es incluido, el intérprete abandona el modo PHP e ingresa al modo HTML al comienzo del archivo objetivo y se reanuda de nuevo al final. Por esta razón, cualquier código en el interior del archivo a incluir que deba ser ejecutado como código PHP, tendrá que ser encerrado dentro de etiquetas de comienzo y fin de bloque PHP.

`include()` devuelve `false` en caso de fallo y emite una advertencia. Si la inclusión tiene éxito, devuelve 1, salvo que desde el archivo incluido se devuelva otro valor. Es decir, es posible ejecutar una sentencia `return` dentro de un archivo incluido con el fin de terminar el procesamiento en ese archivo y volver al script que lo llamó. Además, es posible retornar valores desde los archivos incluidos, los cuales se usarían de la misma forma a como se haría con un valor devuelto por una función normal.

Podemos conseguir lo mismo con la función `require()`, la cual tiene la misma sintaxis y funcionalidad que `include()`, aunque con algunas diferencias. Igual que ocurría con aquél, cuando un fichero es invocado por `require()` esa llamada lo que hace es sustituirse a sí misma por el contenido del fichero especificado. Sin embargo, en caso de fallo producirá un error fatal. En otras palabras, detiene el script mientras que `include()` sólo emitirá una advertencia.

## 4.2 Funciones `include_once()` y `require_once()`

Tanto en el caso de usar la instrucción `include()` como con `require()`, si se intenta incluir dos o más veces un fichero que contenga funciones, se producirá un error (PHP no permite que dos funciones tengan el mismo nombre) y se interrumpirá la ejecución del script. Los errores de ese tipo puede evitarse usando las funciones `include_once()` y `require_once()`, las cuales tienen la misma sintaxis que `include()`. En el caso de que el archivo ya estuviera incluido, no lo volverá a incluir y devolverá `true`.

A diferencia de `include()` y `require()`, van a impedir que un mismo fichero pueda incluirse dos veces.

`include_once()` se puede utilizar en casos donde el mismo fichero podría ser incluido y evaluado más de una vez durante una ejecución particular de un script, así que en este caso, puede ser de ayuda para evitar problemas como la redefinición de funciones, reasignación de valores de variables, etc.

## 5 Proceso de formularios

---

PHP se diseñó para crear aplicaciones web y en este tipo de aplicaciones el usuario se encuentra en el otro extremo de la arquitectura, es decir, en el cliente. La entrada de usuario por tanto no es como una aplicación que se ejecuta localmente, sino que se debe enviar por Internet hasta la aplicación web que reside en un servidor web. La forma de enviar estos datos es mediante el protocolo HTTP.

PHP da soporte completo a una comunicación PHP en la que un usuario introduce unos datos de entrada a través de un formulario HTML y al enviarse se recibe por una aplicación web en un servidor web.

Antes de ver como se procesan los formularios en PHP debemos conocer como se produce la comunicación HTTP entre cliente y servidor, además de algunos elementos PHP que dan soporte a esta comunicación, como los arrays superglobales EGPCS.

### 5.1 El protocolo HTTP

HTTP (*Hypertext Transfer Protocol*) es el protocolo a nivel de aplicación en el modelo de red de Internet que regula el servicio WWW. Este protocolo cliente/servidor establece como los navegadores solicitan recursos a los servidores web y como los servidores responden a estas peticiones. Para entender las técnicas que veremos en este documento necesitamos conocer el funcionamiento básico de HTTP.

#### 5.1.1 Conexión HTTP

A continuación se describe una sesión típica de HTTP entre un navegador web y un servidor web. Las sesiones HTTP están compuestas fundamentalmente por una conexión, una solicitud y una respuesta:

1. El navegador web realiza una petición al servidor web mediante una URL. Esta puede haber sido introducida por el usuario mediante teclado en la barra de dirección del navegador o por otro medio, como haciendo clic en un enlace o accediendo a una URL almacenada en la lista de marcadores.
2. De la URL se obtiene el nombre del dominio del servidor para su traducción por el servicio DNS. El sistema operativo recibe la petición de traducción y realiza una petición de traducción realizando una petición al servidor DNS configurado en la interfaz de red. Esta traducción no forma parte de la comunicación HTTP.
3. Obtenida la dirección del servidor web, el equipo cliente utiliza el protocolo TCP para

establecer una conexión con el mismo, generalmente empleando el puerto 80, que es número de puerto predeterminado para las conexiones HTTP o el puerto 443 para las conexiones HTTPS.

4. Tras conectar con el servidor, el equipo cliente solicita al servidor el recurso web mediante una petición HTTP, la cual está formado por los siguientes campos:

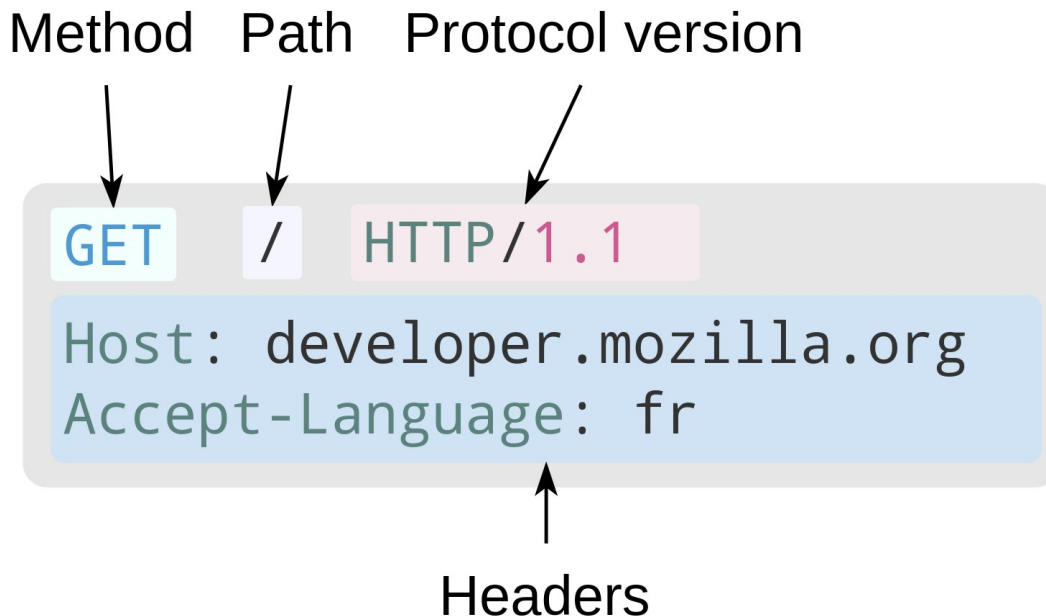


Figura 1: Petición HTTP

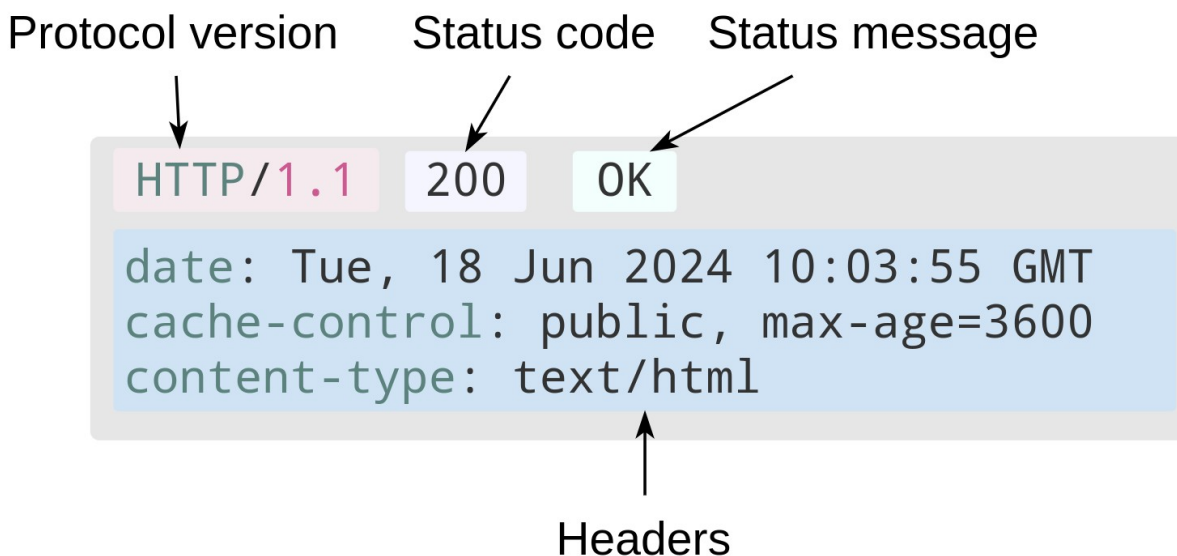
- a) Un método HTTP, normalmente pueden ser un verbo, como: GET, POST o un nombre como: OPTIONS o HEAD, que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o enviar los datos de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones. Algunas de los métodos son:

Método	Acción
GET	Solicitud de un documento a un servidor
HEAD	Solicitud de información sobre un documento, no el documento en sí
POST	Envío de alguna información del cliente al servidor
PUT	Envío de un documento del servidor al cliente
TRACE	Eco de la petición entrante
CONNECT	Reservado
OPTION	Solicitud de algunas opciones disponibles

- b) La URL sin los elementos obvios por el contexto, como pueden ser: el protocolo, el dominio y el puerto.
- c) La versión del protocolo HTTP.



- d) Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.
  - e) O un cuerpo de mensaje, en algún método, como puede ser POST, en el cual envía la información para el servidor.
5. El servidor responde a la solicitud enviando la página web o el recurso solicitado por el cliente. El primero de los paquetes devueltos contiene las cabeceras, que el servidor comunica al cliente. La respuesta está formada por los siguientes campos:



- a) La versión del protocolo HTTP que están usando.
  - b) Un código de estado, indicando si la petición ha tenido éxito, o no, y debido a qué.
  - c) Un mensaje de estado, una breve descripción del código de estado.
  - d) Cabeceras HTTP, como las de las peticiones.
  - e) Opcionalmente, el recurso que se ha pedido, es decir, el comienzo la página web solicitada por el cliente, que muestra el código HTML que da formato a la información para el cliente para que pueda mostrarse en forma de página web.
6. En este momento, si `Keep-Alive` de HTTP está activada en el servidor web, la conexión TCP entre el cliente y el servidor permanece abierta por si el cliente desea solicitar otros recursos al servidor (hojas de estilo, scripts, imágenes, ...). Si `Keep-Alive` está desactivada, la conexión de TCP se cierra una vez descargada la página y se debe establecer una conexión de TCP nueva para poder descargar el archivo siguiente.

Los dos métodos HTTP más habituales son GET y POST. El método GET se diseñó para recuperar información desde el servidor, como un documento, una imagen o los resultados de una consulta a una base de datos. El método POST se diseñó para el envío de información al servidor, como un número de tarjeta de crédito o información para

almacenarse en una base de datos. El método GET es lo que un navegador web utiliza cuando el usuario teclea una URL o hace clic en un enlace. Cuando el usuario envía un formulario se puede utilizar el método GET y POST, especificado por el atributo `method` de la etiqueta `form` de HTML.

### 5.1.2 Formato del mensaje HTTP

Según lo anterior, el formato de los mensajes de petición y respuesta son similares. Ambos se muestran en la figura.



Figura 2.- Mensajes de petición y respuesta

Un mensaje de petición consta de:

- ✓ Una línea de petición → En una misma línea contiene:
  - Método de la petición.- Tal y como se describió anteriormente.
  - URL.- Solamente se incluye el campo path.
  - Versión.- Puede ser 1.1 o 2.
- ✓ Encabezados → Con información que el cliente envía al servidor.
- ✓ Un cuerpo → En algunas ocasiones, depende del método.

Después de las cabeceras la petición contiene una línea en blanco para indicar el final de la sección de encabezado. La petición también puede contener datos adicionales, si es apropiado para el método utilizado, por ejemplo, con el método POST. Si la petición no contiene ningún dato, finaliza con la línea en blanco que hay después de la cabecera.

Cuando el servidor web recibe la petición, la procesa y envía una respuesta. Un mensaje de respuesta consta de:

- ✓ Una línea de estado → Compuesta de:
  - Versión.- Igual que en la petición.

- Código de estado → Este campo se utiliza en el mensaje de respuesta y es un número entero de 3 dígitos. Más abajo se detalla la información de este campo.
- Frase de estado → Una descripción breve del código de estado.
- ✓ Encabezados → Con las mismas cabeceras que envió el cliente más las que añade el servidor.
- ✓ Un cuerpo → No siempre está, pero en las respuestas es más frecuente su presencia.

El código de estado se utiliza en el mensaje de respuesta y su valor está clasificado por grupos y son los siguientes:

- ✓ Los códigos en el rango de 100 sólo son informativos.
- ✓ Del 200 al 299.- Se ha producido una transacción de HTTP con éxito. El código de estado más frecuente es 200 OK.
- ✓ Del 300 al 399.- Ha tenido una redirección a otra URL.
- ✓ Del 400 al 499.- Se ha producido un error. Entre los ejemplos están:
  - ✗ 400 (solicitud incorrecta).- El servidor no puede comprender la sintaxis de la solicitud.
  - ✗ 401 (sin autorización).- Las credenciales del usuario no le permiten iniciar sesión en el servidor.
  - ✗ 403 (prohibido).- Se deniega el acceso por algún motivo ajeno a las credenciales del usuario, como que el cliente tenga una dirección IP restringida o necesite utilizar SSL para tener acceso al servidor.
  - ✗ 404 (no se encuentra el archivo).- El archivo al que se intenta tener acceso no existe en el servidor.
  - ✗ Del 500 al 599.- Se ha producido un error en el servidor o la característica solicitada no está implementada.

### 5.1.3 Los encabezados

Las cabeceras se emplean para intercambiar información entre el cliente y el servidor. Están tanto en la petición como en la respuesta. Por ejemplo, el cliente puede solicitar que el documento sea enviado en un formato especial o el servidor puede enviar información extra sobre el documento. Cada cabecera ocupa una línea y se pueden enviar varias. Cada línea de cabecera tiene un nombre de cabecera, dos puntos, un espacio y un valor de cabecera. Un ejemplo de cabeceras podría ser el siguiente:

```
User-Agent: Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]  
Accept: image/gif, image/jpeg, text/*, */*
```

El encabezado `User-Agent` proporciona información sobre el navegador web, mientras que el encabezado `Accept` especifica los tipos MIME que el navegador acepta.

Por ejemplo:

```
Date: Sat, 29 June 2019 14:07:50 GMT
Server: Apache/2.2.14 (Ubuntu)
Content-Type: text/html
Content-Length: 1845
```

El encabezado `Server` proporciona información sobre el software utilizado como servidor web, mientras que el encabezado `Content-Type` especifica el tipo MIME de los datos incluidos en la respuesta. Después de las cabeceras, la respuesta contiene una línea en blanco seguido de los datos solicitados si la petición tuvo éxito.

## 5.2 Arrays globales EGPCS

La configuración del servidor y la información de la petición, incluyendo los parámetros de formulario y las cookies, son accesibles de tres formas diferentes en los scripts PHP. Generalmente, esta información es referida como EGPCS (acrónimo de *Environment, GET, POST, Cookies y Server*).

PHP crea automáticamente seis arrays globales que contiene la información EGPCS:

- ✓ `$_ENV` → Contiene los valores de las variables de entorno, donde la clave del array es el nombre de la variable de entorno.
- ✓ `$_GET` → Contiene cualquier parámetro que es parte de una petición GET, donde la clave del array es el nombre del parámetro.
- ✓ `$_COOKIE` → Contiene los valores de cookies pasados como parte de la petición, donde la clave del array son los nombres de las cookies.
- ✓ `$_POST` → Contiene cualquier parámetro que es parte de una petición POST, donde la clave del array son los nombres de los parámetros del formulario.
- ✓ `$_SERVER` → Contiene información útil del servidor web. Se verá en detalle en la siguiente sección.
- ✓ `$_FILES` → Contiene información sobre los archivos subidos al servidor.
- ✓ `$_REQUEST` → Es creado por PHP automáticamente y contiene los elementos de los arrays `$_GET`, `$_POST` y `$_COOKIE` en un único array.

Estas variables al ser globales son también visibles dentro de las funciones.

## 5.3 Información del servidor

El array `$_SERVER` contiene mucha información útil sobre el servidor web. Gran parte de ella viene de las variables de entorno requeridas en la especificación CGI. Para saber cuales son sus elementos, su significado y que información contiene cada uno de ellos podemos consultar la documentación oficial de PHP [aquí](#). Algunos de los más habituales son:

- ✓ `PHP_SELF` → El nombre del script actual relativo a la raíz de documentos (por ejemplo `/almacen/carro.php`). Esta variable es útil cuando creamos scripts autoreferenciados, como veremos más adelante.
- ✓ `REQUEST_METHOD` → El método utilizado por el cliente para enviar la información.
- ✓ `PATH_INFO` → Los elementos extra del path proporcionados por el cliente (por ejemplo, `/list/users`).
- ✓ `QUERY_STRING` → Cualquier cosa a la derecha del `?` en la URL. En una petición `GET` de envío de formulario serían los datos del formulario.

El servidor Apache también crea algunas entradas en el array `$_SERVER` para cada cabecera HTTP en la petición. Por cada clave, el nombre de la cabecera se convierte a mayúsculas, el guion medio se convierte en guion bajo, y la cadena "HTTP\_" se añade como prefijo. Por ejemplo, la entrada para la cabecera `User-Agent` tiene la clave `HTTP_USER_AGENT`. Los dos usos más comunes y útiles de las cabeceras son:

- ✓ `HTTP_USER_AGENT` → La cadena del navegador utilizado para identificarlo. Por ejemplo `"Mozilla/5.0 (Windows 2000; U) Opera 6.0 [en]"`.
- ✓ `HTTP_REFERER` → La página del navegador desde la que se solicitó la página actual.

## 5.4 Métodos HTTP

Como vimos antes, hay dos métodos HTTP que un cliente puede utilizar para pasar datos al servidor: `GET` y `POST`. El método que un formulario concreto utiliza se especifica con el atributo `method` de la etiqueta HTML `form`. En teoría, los métodos no son sensibles a la capitalización en HTML, pero en la práctica algunos navegadores requieren el nombre del método en mayúscula.

Un método `GET` codifica los parámetros del formulario dentro de la URL en la cadena de consulta, la cual se indica por el texto que sigue al `?`.

```
/ruta/registro_evento.php?nombre=José&email=jose@gmail.com
```

Una petición `POST` pasa los parámetros del formulario en el cuerpo de la petición HTTP, dejando la URL sin alterar. La diferencia más visible entre `GET` y `POST` es la URL. Ya que todos los parámetros del formulario se codifican en la URL con una petición `GET`, los usuarios pueden sobrescribir, alterar o añadir las consultas `GET`. Esto no puede hacerse con las peticiones `POST`.

La mayor diferencia entre las peticiones `GET` y `POST` es mucho más sutil. La especificación HTTP indica que las peticiones `GET` son idempotentes, es decir, una petición `GET` para una URL concreta, incluyendo los parámetros de formulario, es la misma en dos o más peticiones de esa URL. Así, los navegadores web pueden cachear las páginas de respuesta para las peticiones `GET` ya que la página de respuesta no cambia independientemente de cuantas veces la página se carga. Debido a la idempotencia, las peticiones `GET` deberían ser utilizadas para consultas en las que la página de respuesta no cambia.

Las peticiones `POST` no son idempotentes. Esto significa que no pueden cachearse y por tanto cada petición `POST` genera una respuesta del servidor. Probablemente habremos visto en el navegador web decir *Reenvío de formulario?* Antes de visualizar o recargar cierta página. Esto hace las peticiones `POST` la elección apropiada para consultas cuya página de respuesta pueden cambiar, por ejemplo para visualizar el contenido de un carrito de la compra o la lista de mensajes actuales en un foro.

Es decir, la idempotencia es a menudo ignorada en el mundo real. La caché de los navegadores están generalmente pobremente implementada y el botón *Recarga* es fácil de pulsar. Los programadores tienden a usar `GET` y `POST` simplemente basándose en si quieren mostrar los parámetros en la cadena de consulta de la URL o no. Lo que tenemos que recordar es que las peticiones `GET` no deberían usarse para acciones que requieren un cambio en el servidor, como almacenar un pedido en una base de datos.

El tipo de método utilizado por una petición a una página PHP está disponible a través de `$_SERVER['REQUEST_METHOD']`. Por ejemplo:

```
<?php
...

if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    // gestionar la petición GET
}
else {
    die("Solo se aceptan peticiones GET.");
}
...
?>
```

## 5.5 Parámetros del formulario

Ya hemos visto anteriormente que la entrada de datos del usuario a una aplicación web se produce mediante un formulario. En este epígrafe vamos a ver como una aplicación PHP recibe los valores de los parámetros del formulario. Partimos de un formulario HTML, bien en un documento html o generado por el propio PHP, en el que tenemos un conjunto de controles. Cada control tendrá un valor introducido por el usuario y el nombre del control en su atributo `name` será el que utilicemos en el script PHP para acceder a ese valor.

Supongamos un formulario en el que se registran los usuarios que quieren asistir a un evento con un grupo de personas:


## Formulario de Registro para un Evento


Nombre completo:

Contraseña:

Correo electrónico:

Página web (opcional):

Fecha del evento:  

Hora del evento:  

Número de asistentes:  100

Temas de interés:

- ☐ Tecnología
- ☐ Ciencia
- ☐ Negocios

Participación:

- ☐ Asistente
- ☐ Ponente

País:

Idiomas:

Español

Inglés

Francés

Alemán

Comentarios adicionales:

Figura 3.- Formulario de ejemplo

El formulario anterior comienza indicando el método de la petición en el atributo `method` y el destino de los datos cuando se envíe en el atributo `action`, que es el script

PHP que recoge estos datos y los procesa. Cuando el atributo `action` contiene una URL que comienza por `/` hace referencia a la raíz del propio servidor web.

```
<form action="/registro_eventos.php" method="POST">
...
</form>
```

Cada uno de los controles tendrá su atributo `name` con el nombre del parámetro que emplearemos en el script PHP.

```
<!-- Cuadro de texto -->
<label for="nombre">Nombre completo:</label>
<input type="text" id="nombre" name="nombre" required>

<!-- Cuadro de contraseña -->
<label for="password">Contraseña:</label>
<input type="password" id="password" name="password" required>

<!-- Email -->
<label for="email">Correo electrónico:</label>
<input type="email" id="email" name="email" required>

<!-- URL -->
<label for="website">Página web (opcional):</label>
<input type="url" id="website" name="website">

<!-- Fecha -->
<label for="fecha">Fecha del evento:</label>
<input type="date" id="fecha" name="fecha" required>

<!-- Hora -->
<label for="hora">Hora del evento:</label>
<input type="time" id="hora" name="hora" required>

<!-- Rango numérico -->
<label for="asistentes">Número de asistentes:</label>
<input type="range" id="asistentes" name="asistentes" min="0"
max="500" step="10" value="100">

<!-- Checkbox -->
<label for="temas">Temas de interés:</label>
<div>
  <input type="checkbox" id="tema1" name="temas[]"
value="Tecnología">
  <label for="tema1">Tecnología</label><br>
  <input type="checkbox" id="tema2" name="temas[]"
value="Ciencia">
  <label for="tema2">Ciencia</label><br>
  <input type="checkbox" id="tema3" name="temas[]"
value="Negocios">
  <label for="tema3">Negocios</label>
</div>

<!-- Botones de radio -->
<label for="participacion">Participación:</label>
```



```

<div>
  <input type="radio" id="participar_asistente"
name="participacion" value="Asistente" required>
  <label for="participar_asistente">Asistente</label><br>
  <input type="radio" id="participar_ponente" name="participacion"
value="Ponente">
  <label for="participar_ponente">Ponente</label>
</div>

<!-- Lista de selección única -->
<label for="pais">País:</label>
<select id="pais" name="pais" required>
  <option value="">Selecione...</option>
  <option value="es">España</option>
  <option value="mx">México</option>
  <option value="ar">Argentina</option>
  <option value="cl">Chile</option>
</select>

<!-- Lista de selección múltiple -->
<label for="idiomas">Idiomas:</label>
<select id="idiomas" name="idiomas[]" multiple size="4">
  <option value="espanol">Español</option>
  <option value="ingles">Inglés</option>
  <option value="frances">Francés</option>
  <option value="aleman">Alemán</option>
</select>

<!-- Área de texto -->
<label for="comentarios">Comentarios adicionales:</label>
<textarea id="comentarios" name="comentarios" rows="4"
cols="50"></textarea>

<!-- Botón de envío -->
<input type="submit" value="Enviar">

```

En el script PHP debemos comprobar primero el método de envío y luego utilizamos los array globales que recogen los datos del formulario:

- ✓ Si el método es POST utilizamos el array `$_POST`,
- ✓ Si el método es GET utilizamos el array `$_GET`.
- ✓ Si el formulario incluye un control tipo file tendremos que usar `$_FILES` para acceder a los archivos subidos.

Estos arrays son de tipo asociativo, por tanto, las claves son los nombres de los parámetros y los valores son los valores de los mismos. Ya que los guiones medios son legales en los nombres de campos HTML pero no en los nombres de variables PHP, los guiones medios en los nombres de campo de formulario se convierten a guiones bajos en el array.

El script que recoge los datos del formulario y realiza su procesamiento es el siguiente.

```

<?php
// Verificar si se envió el formulario
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Recoger los datos del formulario
    $nombre = $_POST["nombre"];
    $password = $_POST["password"];
    $email = $_POST["email"];
    $website = isset($_POST["website"]) ? $_POST["website"] : 'No
proporcionado';
    $fecha = $_POST["fecha"];
    $hora = $_POST["hora"];
    $asistentes = $_POST["asistentes"];
    $temas = isset($_POST["temas"]) ? $_POST["temas"] : [];
    $participacion = $_POST["participacion"];
    $pais = $_POST["pais"];
    $idiomas = isset($_POST["idiomas"]) ? $_POST["idiomas"] : [];
    $comentarios = $_POST["comentarios"];

    // Mostrar los datos recogidos
    echo "Datos Recogidos del Formulario:\n\n";
    echo "Nombre completo: $nombre\n";
    echo "Contraseña: (oculta por seguridad)\n";
    echo "Correo electrónico: $email\n";
    echo "Página web: $website\n";
    echo "Fecha del evento: $fecha\n";
    echo "Hora del evento: $hora\n";
    echo "Número estimado de asistentes: $asistentes\n";

    echo "Temas de interés seleccionados: ";
    if (count($temas) > 0) {
        $temas_lista = implode(", ", $temas);
        echo "$temas_lista\n";
    } else {
        echo "No seleccionó temas.\n";
    }

    echo "Preferencia de participación: $participacion\n";
    echo "País: $pais\n";

    echo "Idiomas que hablas: ";
    if (count($idiomas) > 0) {
        $idiomas_lista = implode(", ", $idiomas);
        echo "$idiomas_lista\n";
    } else {
        echo "No seleccionó ningún idioma.\n";
    }

    echo "Comentarios adicionales: $comentarios\n";
} else {
    echo "No se ha enviado ningún formulario.";
}
?>

```

Solamente nos hemos limitado a devolver en la respuesta al cliente los datos enviados para que se presente en el navegador. Dependiendo del escenario de aplicación, el

procesamiento podría ser más complejo e incluir una inserción de los datos en una base de datos.

## 5.6 Páginas autoprocesadas

Un mismo script PHP puede utilizarse para generar un formulario y procesarlo posteriormente. En el ejemplo siguiente, si la página se solicita con el método GET, visualiza un formulario que acepta una temperatura en grados Fahrenheit. Si se invoca con el método POST, entonces la página calcula y visualiza la correspondiente temperatura en grados Celsius.

```
<?php
// Función para convertir de Fahrenheit a Celsius
function fahrenheitToCelsius($fahrenheit) {
    return ($fahrenheit - 32) * 5 / 9;
}

// Verificar el método de la solicitud
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Recoger el valor de la temperatura en Fahrenheit del
    formulario
    $fahrenheit = isset($_POST['fahrenheit']) ?
    floatval($_POST['fahrenheit']) : 0;

    // Realizar la conversión
    $celsius = fahrenheitToCelsius($fahrenheit);

    // Mostrar el resultado de la conversión
    echo "La temperatura $fahrenheit grados Fahrenheit es
    equivalente a " . round($celsius, 2) . " grados Celsius.";

    // Opción para volver a la página del formulario
    echo "<br><br><a href=\"".$_SERVER['PHP_SELF']."\">Volver al
    formulario</a>";
} else {
    // Mostrar el formulario HTML cuando se accede mediante GET
    echo <<<HTML
    <!DOCTYPE html>
    <html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
    initial-scale=1.0">
        <title>Convertir Fahrenheit a Celsius</title>
    </head>
    <body>
        <h2>Convertir Temperatura de Fahrenheit a Celsius</h2>
        <form action="{$_SERVER['PHP_SELF']}" method="post">
            <label for="fahrenheit">Temperatura en
    Fahrenheit:</label>
            <input type="number" id="fahrenheit"
    name="fahrenheit" step="any" required>
            <input type="submit" value="Convertir">
        </form>
```

```

        </body>
    </html>
HTML;
}
?>

```

Otra forma de decidir si visualizar el formulario o procesarlo es comprobar si uno de los parámetros ha sido enviado o no. Esto permite escribir una página autoprocesada que utiliza el método GET para enviar los valores. El siguiente ejemplo muestra una versión modificada de la página de conversión de temperatura que envía los parámetros usando una petición GET. Esta página utiliza la presencia o ausencia de los parámetros para determinar que hacer.

```

<?php
// Función para convertir de Fahrenheit a Celsius
function fahrenheitToCelsius($fahrenheit) {
    return ($fahrenheit - 32) * 5 / 9;
}

// Verificar el método de la solicitud
if ($_SERVER["REQUEST_METHOD"] == "GET") {
    // Comprobar si se ha pasado el parámetro 'fahrenheit'
    if (isset($_GET['fahrenheit']) &&
        is_numeric($_GET['fahrenheit'])) {
        // Recoger el valor de la temperatura
        // en Fahrenheit del formulario
        $fahrenheit = floatval($_GET['fahrenheit']);

        // Realizar la conversión
        $celsius = fahrenheitToCelsius($fahrenheit);

        // Utilizando una variable con el resultado
        // redondeado antes de la interpolación
        $celsiusRedondeado = round($celsius, 2);

        // Mostrar el resultado utilizando interpolación
        echo "La temperatura {$fahrenheit} grados Fahrenheit es
equivalente a {$celsiusRedondeado} grados Celsius.";
    } else {
        // Mostrar el formulario HTML cuando no se ha pasado
        // el parámetro 'fahrenheit'
        echo <<<HTML
        <!DOCTYPE html>
        <html lang="es">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width,
initial-scale=1.0">
            <title>Convertir Fahrenheit a Celsius</title>
        </head>
        <body>
            <h2>Convertir Temperatura de Fahrenheit a
Celsius</h2>
            <form action="{$_SERVER['PHP_SELF']}" method="get">

```

```

        <label for="fahrenheit">Temperatura en
Fahrenheit:</label>
        <input type="number" id="fahrenheit"
name="fahrenheit" step="any" required>
        <input type="submit" value="Convertir">
    </form>
</body>
</html>

HTML;
}
}
?>

```

En el ejemplo anterior utilizamos la función `isset()` para verificar si una variable, o elemento de array, están definidos. También podríamos haber obtenido directamente el valor del parámetro y luego comprobar si es nulo. Sería así

```

<?php
...

if ($_SERVER["REQUEST_METHOD"] == "GET") {
    // Comprobar si se ha pasado el parámetro 'fahrenheit'
    $fahrenheit = $_GET['fahrenheit'];
    if ( is_null($fahrenheit) ) {

        ...

    }
}
?>

```

En el ejemplo, copiamos el valor del parámetro en la variable `$fahrenheit`. Si no se ha enviado el formulario, `$fahrenheit` contiene `null`. Usamos la función `is_null()` para comprobar si debemos mostrar el formulario o procesar los datos.

Por último, en caso de que quisiéramos repetir la conversión una y otra vez de forma cíclica, cada vez que se recibe una petición se realiza la conversión, si se han enviado datos, cosa que no ocurre la primera vez, y posteriormente se presenta el resultado de la conversión. El script quedaría de la siguiente forma:

```

<?php
// Función para convertir de Fahrenheit a Celsius
function fahrenheitToCelsius($fahrenheit) {
    return ($fahrenheit - 32) * 5 / 9;
}

// Verificar el método de la solicitud
if ($_SERVER["REQUEST_METHOD"] == "GET") {
    // Inicializar variable para almacenar el resultado
    $resultado = '';

    // Comprobar si se ha pasado el parámetro 'fahrenheit'
    if (isset($_GET['fahrenheit']) &&

```

```

is_numeric($_GET['fahrenheit'])) {
    // Recoger el valor de la temperatura en Fahrenheit del
    formulario
    $fahrenheit = floatval($_GET['fahrenheit']);

    // Realizar la conversión
    $celsius = fahrenheitToCelsius($fahrenheit);

    // Utilizando una variable con el resultado redondeado
    antes de la interpolación
    $celsiusRedondeado = round($celsius, 2);

    // Preparar el resultado
    $resultado = "La temperatura {$fahrenheit} grados
    Fahrenheit es equivalente a {$celsiusRedondeado} grados
    Celsius.";
}

// Mostrar el resultado y el formulario HTML
echo <<<HTML
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Convertir Fahrenheit a Celsius</title>
</head>
<body>
    <h2>Convertir Temperatura de Fahrenheit a Celsius</h2>
    <form action="{$_SERVER['PHP_SELF']}" method="get">
        <label for="fahrenheit">Temperatura en
Fahrenheit:</label>
        <input type="number" id="fahrenheit"
name="fahrenheit" step="any">
        <input type="submit" value="Convertir">
    </form>
    <p>$resultado</p>
</body>
</html>
HTML;
}
?>

```

## 5.7 Sticky forms

Muchos sitios web utilizan una técnica conocida como *sticky forms* en la cual los resultados de una consulta se acompañan por un formulario de búsqueda cuyo valor por defecto es aquél de la consulta previa. Por ejemplo, si buscamos en Google *Programming PHP* en la parte superior de la página de resultados contiene otro cuadro de texto para una nueva búsqueda, el cual ya contiene *Programming PHP*. Para refinar la búsqueda a *Programming PHP Web Services* podemos simplemente añadir el texto extra.

Este comportamiento es fácil de implementar. El siguiente ejemplo muestra el script de

conversión de temperaturas hecho con esta técnica la cual consiste en utilizar el valor enviado en el formulario como valor por defecto para crear el campo HTML.

```
<?php
// Función para convertir de Fahrenheit a Celsius
function fahrenheitToCelsius($fahrenheit) {
    return ($fahrenheit - 32) * 5 / 9;
}

// Verificar el método de la solicitud
if ($_SERVER["REQUEST_METHOD"] == "GET") {
    // Inicializar variable para almacenar el resultado
    $resultado = '';
    $fahrenheitValor = '';

    // Comprobar si se ha pasado el parámetro 'fahrenheit'
    if (isset($_GET['fahrenheit']) &&
    is_numeric($_GET['fahrenheit'])) {
        // Recoger el valor de la temperatura en Fahrenheit
        // del formulario
        $fahrenheit = floatval($_GET['fahrenheit']);

        // Guardar el valor para mostrarlo en el formulario
        $fahrenheitValor = $fahrenheit;

        // Realizar la conversión
        $celsius = fahrenheitToCelsius($fahrenheit);

        // Utilizando una variable con el resultado
        // redondeado antes de la interpolación
        $celsiusRedondeado = round($celsius, 2);

        // Preparar el resultado
        $resultado = "La temperatura {$fahrenheit} grados
Fahrenheit es equivalente a {$celsiusRedondeado} grados
Celsius.";
    }

    // Mostrar el resultado y el formulario HTML
    echo <<<HTML
    <!DOCTYPE html>
    <html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Convertir Fahrenheit a Celsius</title>
    </head>
    <body>
        <h2>Convertir Temperatura de Fahrenheit a Celsius</h2>
        <form action="{$_SERVER['PHP_SELF']}" method="get">
            <label for="fahrenheit">Temperatura en
Fahrenheit:</label>
            <input type="number" id="fahrenheit"
name="fahrenheit" step="any" value="{ $fahrenheitValor }">
```

```

        <input type="submit" value="Convertir">
    </form>
    <p>$resultado</p>
</body>
</html>
HTML;
}
?>

```

## 5.8 Parámetros multivaluados

Las listas de selección HTML, creadas con la etiqueta `select`, permiten selecciones múltiples. Para asegurar que PHP reconoce múltiples valores que el navegador le va a pasar en un parámetro multivaluado necesitamos utilizar los corchetes `[]` después del nombre del campo en el formulario HTML. Por ejemplo:

```

<select name="lenguajes[]" size="4" multiple>
  <option name="c">C</option>
  <option name="c++">C++</option>
  <option name="php">PHP</option>
  <option name="perl">Perl</option>
  <option name="python">Python</option>
  <option name="java">Java</option>
</select>

```

Ahora, cuando el usuario envía el formulario, `$_GET['lenguajes']` contiene un array en lugar de una simple cadena. Este array contiene los valores que fueron seleccionados por el usuario. En el siguiente ejemplo el formulario proporciona un lenguajes de programación. Cuando se envía, devuelve un resumen de los conocimientos del usuario.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Lenguajes de programación</title>
</head>
<body>

<form action="<?=$_SERVER['PHP_SELF']; ?>" method="GET">
Selecciona los lenguajes de programación que dominas:<br>
<select name="lenguajes[]" size="4" multiple>
  <option name="c">C</option>
  <option name="c++">C++</option>
  <option name="php">PHP</option>
  <option name="perl">Perl</option>
  <option name="python">Python</option>
  <option name="java">Java</option>
</select>
<br>

<input type="submit" name="operacion" value="Enviar">

```



```

</form>

<?php
if (array_key_exists('operacion', $_GET)) {
    $descripcion = join(', ', $_GET['lenguajes']);
    echo "Dominas los siguientes lenguajes: {$descripcion}";
}
?>
</body>
</html>

```

El botón de envío de formulario tiene un nombre de parámetro. Comprobamos la presencia o ausencia de este valor de parámetro de una forma diferente a como lo hemos hecho antes. En este caso comprobamos si en un array asociativo, como `$_GET`, contiene la clave con el nombre del parámetro asignado al botón de envío.

La misma técnica se aplica para cualquier campo de formulario donde se pueden devolver múltiples valores. El siguiente ejemplo revisa el anterior donde se emplean casillas de verificación en lugar de una lista de selección. Nótese que solo ha cambiado el código HTML, pero el código que procesa el formulario no necesita saber si los valores múltiples vienen de casillas de verificación de la lista de selección.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Lenguajes de programación</title>
</head>
<body>

<form action="<?=$_SERVER['PHP_SELF']; ?>" method="GET">
Selecciona los lenguajes de programación que dominas:<br>
<input type="checkbox" name="lenguajes[]" value="c"> C<br>
<input type="checkbox" name="lenguajes[]" value="c++"> C++<br>
<input type="checkbox" name="lenguajes[]" value="php"> PHP<br>
<input type="checkbox" name="lenguajes[]" value="perl"> Perl<br>
<input type="checkbox" name="lenguajes[]" value="python">
Python<br>
<input type="checkbox" name="lenguajes[]" value="java"> Java<br>
<br>
<input type="submit" name="operacion" value="Enviar">
</form>

<?php
if (array_key_exists('operacion', $_GET)) {
    $descripcion = join(', ', $_GET['lenguajes']);
    echo "Dominas los siguientes lenguajes: {$descripcion}";
}
?>
</body>
</html>

```

## 5.9 Parámetros multivaluados sticky

Es posible hacer selecciones múltiples con persistencia de su valor. Necesitamos comprobar si cada valor posible en el formulario ha sido enviado previamente, si es así se añade el atributo `checked` al elemento `input`, con lo que la casilla de verificación estaría marcada. Si no se envió previamente, el elemento `input` no tendrá la casilla marcada. Por ejemplo

```
<input type="checkbox" name="lenguajes[]" value="c"
<?=in_array('c', $_GET['lenguajes']) ? "checked" : ""?>
<br>
```

Podemos usar esta técnica para cada casilla de verificación, pero es repetitivo y propenso a errores. Es más fácil escribir una función que genera el código HTML para los posibles valores y trabaja con una copia de los parámetros enviados. El siguiente ejemplo muestra una nueva versión de la selección múltiple con casillas de verificación. Aunque este formulario es igual al del ejemplo previo, entre bastidores hay cambios sustanciales en cuanto a la forma de generarse.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Lenguajes de programación</title>
</head>
<body>

<?php
// Creamos una copia de los valores
// del formulario si hay alguno
$attrs = $_GET['lenguajes'];
if (!is_array($attrs)) {
    $attrs = array();
}

// Función para identificar las casillas con nombre
function makeCheckboxes($name, $query, $options) {
    foreach ($options as $value => $label) {
        $checked = in_array($value, $query) ? "checked" : '';
        echo "<input type=\"checkbox\" name=\"{$name}\""
            value=\"{$value}\" {$checked}>";
        echo "{$label}<br>\n";
    }
}

// La lista de valores y etiquetas para las casillas
$lenguajes = array(
    'c' => "C",
    'c++' => "C++",
    'php' => "PHP",
    'perl' => "Perl",
```

```

    'python' => "Python",
    'java' => "Java"
);
?>

<form action="<?=$_SERVER['PHP_SELF']; ?>" method="GET">
Selecciona los lenguajes de programación que dominas:<br>

<?php
makeCheckboxes('lenguajes[]', $attrs, $lenguajes);
?>

<br>
<input type="submit" name="operacion" value="Enviar">
</form>

<?php
if (array_key_exists('operacion', $_GET)) {
    $descripcion = join(', ', $_GET['lenguajes']);
    echo "Dominas los siguientes lenguajes: {$descripcion}";
}
?>
</body>
</html>

```

La función `makeCheckboxes()` tiene tres parámetros:

- ✓ El nombre del grupo de casillas de verificación. Este nombre será común a todas las casillas.
- ✓ El array de valores por defecto. Consiste en la copia de `$_GET` que hicimos al principio y donde aparecen las opciones que se han enviado en la petición actual.
- ✓ El array que traduce valores a descripciones. La lista de opciones para las casillas de verificación está en el array `$lenguajes`. La clave será el atributo `value` de la casilla y el valor será la etiqueta que acompañe a la casilla.

## 5.10 Subida de archivos

Para gestionar la subida de archivo utilizamos el array `$_FILES`. Utilizando las diferentes funciones de autenticación y subida de archivos podemos controlar quién está autorizado para subir archivos y que hacer con ellos una vez están en el sistema. El siguiente código muestra un formulario que permite subidas de archivos en la misma página.

```

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Enviar Curriculum</title>
</head>
<body>

```

```
<h1>Formulario de Envío de Curriculum</h1>
<form action="procesar_cv.php" method="POST"
enctype="multipart/form-data">
  <input type="hidden" name="MAX_FILE_SIZE" value="10240">
  <label for="dni">DNI:</label><br>
  <input type="text" id="dni" name="dni" required><br><br>

  <label for="nombre">Nombre:</label><br>
  <input type="text" id="nombre" name="nombre" required><br><br>

  <label for="cv">Curriculum Vitae (PDF):</label><br>
  <input type="file" id="cv" name="cv" accept="application/pdf"
required><br><br>

  <button type="submit">Enviar</button>
</form>
</body>
</html>
```

El mayor problema con la subida de archivos es el riesgo de obtener un archivo que es demasiado grande de procesar. PHP tiene dos formas de prevenir esto: usar un límite rígido y un límite blando. La opción `upload_max_filesize` en `php.ini` proporciona un límite rígido del tamaño del archivo subido (por defecto es 2 MB). Si el formulario incluye un parámetro llamado `MAX_FILE_SIZE` antes del campo con el archivo a subir, PHP utiliza ese valor como límite blando. Por ejemplo, en el ejemplo previo el límite se ha establecido en 10 KB. PHP ignora este parámetro si está establecido a un valor mayor que `upload_max_filesize`. También, nótese que la etiqueta del formulario tiene el atributo `enctype` con el valor `multipart/form-data`.

El atributo `accept` en el elemento `input` de tipo `file` especifica un filtro para los tipos de archivo que el usuario puede introducir en el cuadro de diálogo que selecciona el archivo del equipo local. Este atributo solo puede usarse en elementos `input` de tipo `file` y no debe usarse como una herramienta de validación, sino que los archivos subidos se validan en el servidor.

Cuando la petición llega al servidor, existe un array global que contiene los archivos subidos (puede ser más de uno). Este array es `$_FILES` y cada elemento en este array es, a su vez, otro array que proporciona información del archivo subido. Las claves de este array son:

- ✓ `name` → El nombre del archivo subido. Es posible encontrar diferentes convenciones de nombres entre el cliente y el servidor. Por ejemplo, el path `D:\PHOTOS\ME.JPG` en un cliente Windows podría no tener significado para un servidor Linux.
- ✓ `type` → El tipo MIME del archivo subido. Fundamental si se están subiendo archivos de varios tipos ya que es muy conveniente conocer de qué tipo es.
- ✓ `size` → El tamaño del archivo subido en bytes. Si el usuario intenta subir un archivo demasiado largo, será 0.
- ✓ `tmp_name` → El nombre del archivo temporal en el servidor que contiene el archivo

subido. Si el cliente intenta subir un archivo demasiado grande, su valor es none.

El siguiente script PHP sería el necesario para gestionar los archivos subidos en el anterior código HTML.

```
<?php
// Verificar si el formulario fue enviado
if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    // Verificar si el archivo fue cargado correctamente
    if (isset($_FILES['cv']) &&
        $_FILES['cv']['error'] === UPLOAD_ERR_OK) {

        // Recoger datos del formulario
        $dni = $_POST['dni'];
        $nombre = $_POST['nombre'];
        $archivo = $_FILES['cv'];

        // Ruta del directorio donde se guardarán los archivos
        $directorio = 'curriculums/';

        // Verificar que el directorio existe, sino crearlo
        if (!is_dir($directorio)) {
            mkdir($directorio, 0777, true);
        }

        // Obtener la extensión del archivo
        $extension = pathinfo($archivo['name'],
PATHINFO_EXTENSION);

        // Nombre único para el archivo
        //(puedes personalizarlo, aquí usamos el
        // DNI como nombre de archivo)
        $nombreArchivo = $dni . '.' . $extension;

        // Ruta completa donde se guardará el archivo
        $rutaDestino = $directorio . $nombreArchivo;

        // Mover el archivo cargado a la carpeta de destino
        if (move_uploaded_file($archivo['tmp_name'],
$rutaDestino)) {
            echo "El archivo se ha subido correctamente.";
        } else {
            echo "Hubo un error al subir el archivo.";
        }
    } else {
        echo "No se ha subido ningún archivo o ha ocurrido un
error.";
    }
} else {
    echo "Formulario no enviado.";
}
?>
```

Veamos en detalle el script anterior:

Hemos verificado la subida del archivo comprobando si existe un elemento en el array `$_FILES`. La clave del array es el valor del atributo `name` en el elemento `<input type='file'>` del formulario HTML. Además, comprobamos que no ha habido error en la subida con el elemento `error` del array asociativo de cada archivo en `$_FILES`. Si el valor es `UPLOAD_ERR_OK` es que la subida se ha producido correctamente.

Otra forma correcta de comprobar si un fichero fue subido con éxito es utilizar la función `is_uploaded_file()` de la siguiente forma:

```
if (is_uploaded_file($_FILES['cv']['tmp_name'])) {  
    // Si devuelve True, el archivo ha sido subido  
}
```

Devuelve `true` si el archivo pasado como argumento fue subido mediante HTTP POST. Hay que indicar el nombre temporal, no sirve el nombre del archivo.

El directorio donde se van a almacenar los archivos subidos lo podemos elegir e incluso crear si no lo estuviera ya. Lo creamos con la función `mkdir()`. Este directorio es especialmente importante que tenga permisos de escritura para el usuario que ejecuta el servidor web Apache. Si no fuera así, al intentar guardarlo dará error por no estar autorizado para crear archivos.

Con la función `pathinfo()` podemos obtener la extensión del archivo. La usamos para formar el nombre del archivo que vamos a guardar. Empleamos el dni enviado junto con la extensión obtenida para formar este nombre de archivo.

Los ficheros se almacenan en el directorio de archivos temporales por defecto del servidor el cual se especifica en `php.ini` con la opción `upload_tmp_dir`. Para mover el archivo a un directorio diferente utilizamos la función `move_uploaded_file()`. Esta función recibe dos argumentos, el archivo temporal subido y la ruta completa donde se va a guardar, incluido su nombre.

La invocación de `move_uploaded_file()` automáticamente comprueba si el archivo fue subido. Cuando el script finaliza, cualquier archivo subido a ese script es borrado del directorio temporal.

## 5.11 Saneamiento de valores de parámetros

La función `htmlspecialchars()` en PHP se utiliza para convertir caracteres especiales en entidades HTML, lo que es útil para prevenir ataques de inyección de código (XSS) al mostrar contenido en una página web. Esta función se utiliza principalmente para desinfectar el contenido que el usuario ha introducido antes de mostrarlo en HTML, evitando que ciertos caracteres se interpreten como código HTML.

La sintaxis de la función es:

```
htmlspecialchars(string $string,  
                 int $flags = ENT_COMPAT | ENT_HTML401,  
                 ?string $encoding = null,  
                 bool $double_encode = true): string
```

Los parámetros son los siguientes:

- ✓ `$string` → La cadena de texto que a convertir. Por ejemplo, el contenido de un campo de formulario o cualquier entrada del usuario.
- ✓ `$flags` → (opcional) Indicador que determina cómo deben manejarse las comillas y otras entidades especiales. Algunos de los valores más comunes son:
  - `ENT_COMPAT` (por defecto): Convierte solo las comillas dobles (`"`), no las simples (`'`).
  - `ENT_QUOTES`: Convierte tanto las comillas dobles como las simples.
  - `ENT_NOQUOTES`: No convierte ninguna comilla.
  - `ENT_HTML401`, `ENT_HTML5`, `ENT_XML1`, `ENT_XHTML`: Opciones que especifican el estándar de salida para las entidades HTML.
- ✓ `$encoding` → (opcional) Especifica el tipo de codificación de caracteres. Si no se proporciona, por defecto se usa el valor de `default_charset`. Algunos ejemplos de codificación son: UTF-8, ISO-8859-1, ASCII
- ✓ `$double_encode` → (opcional): Un valor booleano que especifica si los caracteres que ya están codificados como entidades HTML deben ser codificados de nuevo. Si se establece en `false`, evitará que las entidades ya codificadas como `&amp;` se vuelvan a codificar como `&amp;`. Por defecto, este valor es `true`.

Cuando se usan caracteres especiales como `<`, `>`, `&`, `"` y `'` dentro de un documento HTML, estos caracteres pueden ser interpretados como parte del código HTML, lo que puede permitir la ejecución de código no deseado o malicioso. La función `htmlspecialchars()` convierte esos caracteres especiales en sus entidades correspondientes:

- ✓ `<` se convierte en `&lt;`;
- ✓ `>` se convierte en `&gt;`;
- ✓ `&` se convierte en `&amp;`;
- ✓ `"` se convierte en `&quot;`;
- ✓ `'` (si se especifica con la opción `ENT_QUOTES`) se convierte en `&#039;`;

Esto asegura que los caracteres sean tratados como texto y no como código HTML, previniendo posibles ataques de inyección de código.

Veamos un ejemplo básico

```
<?php
$input = "<script>alert('XSS');</script>";
echo htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
?>
```

Salida que produciría es:

```
<script>alert('XSS');</script>
```

En este ejemplo, el código malicioso que intenta ejecutar un script se convierte en texto plano, lo que evita que el navegador lo ejecute.

Es común usar `htmlspecialchars()` al mostrar datos que han sido enviados por los usuarios, como en formularios, comentarios o cualquier entrada de datos. De esta manera, los caracteres peligrosos se convierten en texto inofensivo en lugar de ejecutarse como código.

Por ejemplo, en un script anterior que recogía los datos de un registro de eventos accedíamos directamente al array `$_POST`, con lo que existe la posibilidad de que contengan código HTML malicioso. Para evitarlo, limpiamos los datos con `htmlspecialchars()`.

```
<?php
// Verificar si se envió el formulario
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    // Recoger los datos del formulario
    $nombre = htmlspecialchars($_POST["nombre"]);
    $password = htmlspecialchars($_POST["password"]);
    $email = htmlspecialchars($_POST["email"]);
    $website = isset($_POST["website"]) ?
htmlspecialchars($_POST["website"]) : 'No proporcionado';
    $fecha = htmlspecialchars($_POST["fecha"]);
    $hora = htmlspecialchars($_POST["hora"]);
    $asistentes = htmlspecialchars($_POST["asistentes"]);

    // Se recoge un array de valores.
    // No se usa htmlspecialchars()
    $temas = isset($_POST["temas"]) ? $_POST["temas"] : [];
    $participacion = htmlspecialchars($_POST["participacion"]);
    $pais = htmlspecialchars($_POST["pais"]);

    // Igual que antes con temas
    $idiomas = isset($_POST["idiomas"]) ? $_POST["idiomas"] : [];

    $comentarios = htmlspecialchars($_POST["comentarios"]);

    // Mostrar los datos recogidos
    echo "Datos Recogidos del Formulario:\n\n";
    echo "Nombre completo: $nombre\n";
    echo "Contraseña: (oculta por seguridad)\n";
    echo "Correo electrónico: $email\n";
    echo "Página web: $website\n";
    echo "Fecha del evento: $fecha\n";
    echo "Hora del evento: $hora\n";
    echo "Número estimado de asistentes: $asistentes\n";

    echo "Temas de interés seleccionados: ";
    if (count($temas) > 0) {
        $temas_lista = implode(", ", $temas);
        echo "$temas_lista\n";
    }
}
```



```

    } else {
        echo "No seleccionó temas.\n";
    }

    echo "Preferencia de participación: $participacion\n";
    echo "País: $pais\n";

    echo "Idiomas que hablas: ";
    if (count($idiomas) > 0) {
        $idiomas_lista = implode(", ", $idiomas);
        echo "$idiomas_lista\n";
    } else {
        echo "No seleccionó ningún idioma.\n";
    }

    echo "Comentarios adicionales: $comentarios\n";
} else {
    echo "No se ha enviado ningún formulario.";
}

```

Nótese que los campos de formulario que son arrays no se limpian con `htmlspecialchars()`. En este caso es mejor validarlos como se verá en el siguiente epígrafe. Si un usuario introduce algo como:

```
<script>alert('Hola');</script>
```

En lugar de ejecutarse el código, se mostrará como:

```
Hola, &lt;script&gt;alert(&#039;Hola&#039;);&lt;/script&gt;
```

Los beneficios que proporciona `htmlspecialchars()` son:

- ✓ Prevención de XSS: Protege contra la ejecución de scripts maliciosos inyectados en las páginas web mediante datos introducidos por el usuario.
- ✓ Seguridad: Ayuda a que los datos del usuario se traten como contenido de texto y no como código ejecutable.
- ✓ Sencillez: Es fácil de usar y eficiente para la mayoría de los casos en los que se quiere desinfectar una entrada del usuario antes de mostrarla en HTML.

Es recomendable usar `htmlspecialchars()` siempre que:

- ✓ Se muestren datos introducidos por el usuario en una página web.
- ✓ Se quiera evitar que caracteres como `<`, `>`, y `&` sean interpretados como parte de la sintaxis HTML.
- ✓ Se manipule cualquier tipo de entrada que potencialmente podría ser maliciosa.

De este modo, se garantiza que los datos mostrados en el navegador sean tratados como texto seguro.

## 5.12 Validar el formato de los valores de parámetros

Cuando recibimos datos de un formulario existe la posibilidad de que su formato no sea adecuado. Gracias al uso de diferentes tipos de controles de formulario, gran parte de este trabajo lo realiza el cliente. Por ejemplo, si necesitamos un dato que sea una dirección de correo electrónico, no podemos usar un control de tipo `input`, ya que sería un cuadro de texto donde se introduciría cualquier cadena de caracteres, y lo que se necesita es una cadena cuyo formato se ajuste a una dirección de correo electrónico. De ahí que resulte más conveniente utilizar un control de tipo `email`.

También, disponemos del atributo `pattern` en los controles de formulario que admiten una entrada de texto, el cual nos permite incluir una expresión regular a la que el texto introducido por el usuario se tiene que ajustar para poder enviarse el formulario.

Sin embargo, lo anterior proporciona validación del lado del cliente (navegador), pero no es un sustituto de la validación del lado del servidor, ya que los datos pueden ser modificados en el lado del cliente. Es importante validar los datos también en el servidor para mayor seguridad.

Para validar los datos recibidos con un formulario contamos con la función `filter_input()` la cual toma una variable global y la filtra. Su sintaxis es la siguiente:

```
filter_input(  
    int $type,  
    string $variable_name,  
    int $filter = FILTER_DEFAULT,  
    mixed $options = ?  
) : mixed
```

El significado de cada argumento es:

- ✓ `type` → Indica donde se encuentra la variable global a filtrar. Puede ser una de las siguientes constantes: `INPUT_GET`, `INPUT_POST`, `INPUT_COOKIE`, `INPUT_SERVER` o `INPUT_ENV`.
- ✓ `variable_name` → Nombre de la variable a filtrar. Será la clave con la que se accede a cada variable global. Será por tanto el nombre del control del formulario para GET o POST, el nombre de la COOKIE, la variable de servidor o de entorno.
- ✓ `filter` → El identificador del filtro a aplicar. La página [tipos de filtros](#) del manual oficial de PHP enumera los filtros disponibles.
- ✓ `options` → Array asociativo de opciones o disyunción lógica de flags. Los flags se encuentran en la lista de filtros disponibles.

En caso de éxito, se devuelve el valor de la variable pedida, `false` si el filtro falla o `null` si la variable `variable_name` no está definida. Si se usa el flag `FILTER_NULL_ON_FAILURE`, devuelve `false` si la variable no está definida y `null` si el filtro falla.

Veamos un ejemplo. El siguiente formulario recoge los datos de una empresa, en los

que hay que introducir cuantos trabajadores tiene, su capital en millones de €, la dirección de correo, el nombre de dominio y finalmente la URL de la empresa. Todos estos datos tienen un formato que tenemos que validar para garantizarnos de que son válidos y, por ende, no suponen un riesgo de error en nuestra aplicación si los presentamos en alguna web.

```
<?php
// Definir variables para almacenar errores y resultados
$mensajesError = [];
$mensajeOK = "";

// Validar el formulario al ser enviado
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validar número de empleados (entero)
    $num_empleados = filter_input(INPUT_POST, 'num_empleados',
    FILTER_VALIDATE_INT);
    if ($num_empleados === false || $num_empleados <= 0) {
        $mensajesError['num_empleados'] = "Por favor, introduzca
un número entero válido para el número de empleados.";
    }

    // Validar capital en millones (coma flotante)
    $capital = filter_input(INPUT_POST, 'capital',
    FILTER_VALIDATE_FLOAT);
    if ($capital === false || $capital < 0) {
        $mensajesError['capital'] = "Por favor, introduzca un
valor válido para el capital en millones.";
    }

    // Validar correo electrónico
    $email = filter_input(INPUT_POST, 'email',
    FILTER_VALIDATE_EMAIL);
    if ($email === false) {
        $mensajesError['email'] = "Por favor, introduzca una
dirección de correo válida.";
    }

    // Validar nombre de dominio de empresa
    $dominio = filter_input(INPUT_POST, 'dominio',
    FILTER_VALIDATE_DOMAIN, FILTER_FLAG_HOSTNAME);
    if ($dominio === false) {
        $errorMessages['dominio'] = "Por favor, introduzca un
nombre de dominio válido.";
    }

    // Validar URL del sitio web de la empresa
    $url = filter_input(INPUT_POST, 'url', FILTER_VALIDATE_URL);
    if ($url === false) {
        $mensajesError['url'] = "Por favor, introduzca una URL
válida.";
    }

    // Si no hay errores, mostrar OK
    if (empty($mensajesError)) {
```

```

        $mensajeOK = "Todos los datos se han validado
correctamente.";
    }
}
?>

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Formulario con Validación PHP</title>
    <style type="text/css">
        .ok {
            color:green;
        }
        .error {
            color:red;
        }
    </style>
</head>
<body>
    <h2>Formulario de Validación</h2>

    <?php
        if (!empty($successMessage)) {
            echo "<p class='ok'>$mensajeOK</p>";
        }
    ?>

    <form method="POST" action="">
        <label for="num_empleados">Número de Empleados:</label>
        <input type="text" name="num_empleados"
id="num_empleados" value="<?="
htmlspecialchars($_POST['num_empleados'] ?? '') ?>">
        <span class="error"><?=" $errorMessages['num_empleados']
?? '' ?></span>
        <br><br>

        <label for="capital">Capital de la Empresa (en
millones):</label>
        <input type="text" name="capital" id="capital" value="<?="
htmlspecialchars($_POST['capital'] ?? '') ?>">
        <span class="error"><?=" $errorMessages['capital'] ?? '' ?
></span>
        <br><br>

        <label for="email">Correo Electrónico de
Contacto:</label>
        <input type="email" name="email" id="email" value="<?="
htmlspecialchars($_POST['email'] ?? '') ?>">
        <span class="error"><?=" $errorMessages['email'] ?? '' ?
></span>
        <br><br>

```

```

        <label for="domain">Dominio de la Empresa:</label>
        <input type="text" name="domain" id="domain" value="<?=
htmlspecialchars($_POST['domain']) ?>">
        <span class="error"><?= $errorMessagees['domain'] ?? ' ' ?
></span>
        <br><br>

        <label for="url">Sitio Web de la Empresa (URL):</label>
        <input type="url" name="url" id="url" value="<?=
htmlspecialchars($_POST['url']) ?>">
        <span class="error"><?= $errorMessagees['url'] ?? ' ' ?
></span>
        <br><br>

        <input type="submit" value="Enviar">
    </form>
</body>
</html>

```

### 5.13 Validación de valores de parámetros

En el epígrafe anterior vimos como validar el formato de los datos del formulario. Esto indica que solo admitimos en nuestro script datos que se ajusten a un determinado formato, pero no el valor del dato en sí.

Una de las cosas más importantes que necesitamos entender es que cuando desarrollamos un sitio web seguro, toda la información no generada dentro de la aplicación está potencialmente contaminada, o al menos sospechosa. Esto incluye datos de formularios, ficheros y bases de datos.

Cuando los datos se describen como contaminados, no necesariamente significan que son maliciosos. Significa que podrían ser maliciosos. No podemos confiar en el origen de los datos, por tanto deberíamos inspeccionarlos para asegurarnos que son válidos. Este proceso de inspección se conoce como filtrado y simplemente queremos permitir la entrada a la aplicación de datos válidos.

Hay algunas buenas prácticas para el proceso de filtrado:

- ✓ Usar un enfoque de lista blanca → Esto significa que asumimos que el dato es inválido a menos que podamos probar que es válido.
- ✓ Nunca corregir datos inválidos → Intentar corregir un dato inválido frecuentemente lleva a vulnerabilidades de seguridad debido a errores.
- ✓ Usar una convención de nombres para distinguir entre dato filtrado y contaminado. El filtrado es inútil si no podemos determinar si algo ha sido filtrado.

Pensemos en el siguiente código HTML.

```

<form action="procesar_color.php" method="POST">
<p>Selecciona un color:

```

```
<select name="color">
<option value="rojo">rojo</option>
<option value="verde">verde</option>
<option value="azul">azul</option>
</select>
<input type="submit" value="Enviar"/></p>
</form>
```

Aparentemente podríamos confiar en que el valor de `$_POST['color']` en `procesa_color.php`. El uso de una lista de selección única en el formulario restringe el valor que puede introducir el usuario. Sin embargo, las peticiones HTTP no tienen restricciones en los campos que contienen y la validación del lado del cliente no es suficiente. Hay numerosas formas maliciosas de enviar datos a la aplicación y la única defensa es no confiar y filtrar la entrada de datos.

Siguiendo con el ejemplo anterior podría ser así:

```
<?php
$filtrado = array();
switch($_POST['color']) {
    case 'rojo':
    case 'verde':
    case 'azul':
        $filtrado['color'] = $_POST['color'];
        break;
    default:
        /* Aquí se gestiona el error */
        break;
}
?>
```

Este ejemplo demuestra una simple convención de nombres. Inicializamos un array asociativo llamado `$filtrado`. Para cada dato de entrada, validamos su valor y lo almacenamos en el array usando el nombre del campo como clave. Esto reduce la posibilidad de que datos contaminados estén por error en los datos filtrados, ya que siempre consideraremos que los datos que no están en el array están contaminados.

La lógica de filtrado depende del dato que estemos inspeccionando y, cuanto más restrictivos podamos ser, mejor. Por ejemplo, consideremos un formulario de registro que recoge el nombre de un usuario. Evidentemente, hay muchas posibilidades de nombres de usuario, pero el mejor enfoque es permitir exclusivamente letras (mayúsculas y minúsculas) y dígitos numéricos. Nuestra lógica de filtrado podría ser la siguiente:

```
<?php
...
$filtrado= array();

if (ctype_alnum($_POST['usuario'])) {
    $filtrado['usuario'] = $_POST['usuario'];
}
else {
    /* Gestionamos aquí el error */
}
```

```
}  
...  
?>
```

La función `ctype_alnum()` verifica que todos los caracteres del argumento pasado son letras o números. No es necesario sanear la variable con `htmlspecialchars()` ya que la propia función anterior verificaría que no tuviera elementos HTML.

Tal vez con lo anterior no es suficiente. Si queremos además, que el nombre del usuario tenga una longitud mínima y máxima, utilizamos la función `mb_strlen()`.

```
<?php  
...  
$filtrado= array();  
  
$longitud = mb_strlen($_POST['usuario']);  
  
if (ctype_alnum($_POST['usuario'])  
    AND ($longitud > 0)  
    AND ($longitud <= 32) ) {  
    $filtrado['usuario'] = $_POST['usuario'];  
}  
else {  
    /* Gestionamos aquí el error */  
}  
...  
?>
```

Es posible que los caracteres que queremos permitir pertenezcan a un grupo, como los alfanuméricos y algún símbolo, y es aquí donde las expresiones regulares y la función `preg_match()` nos pueden ayudar.

## 6 Bibliografía

---

TATROE, K y MACYNTYRE, P., Programming PHP 4th Edition 2020 O'Reilly Media, Inc., ISBN 978-1-492-05413-9