

Generación web con PHP

Este obra está bajo una [licencia de Creative Commons Reconocimiento-CompartirIgual 4.0 Internacional](https://creativecommons.org/licenses/by-sa/4.0/).



Contenidos

Introducción.....	5
Instalación del entorno y herramientas de programación.....	6
Formularios HTML.....	7
Método de envío: GET o POST.....	7
Controles del formulario.....	8
Lenguaje PHP.....	10
Variables.....	11
Generación de HTML.....	11
Evaluación de cadenas.....	11
Comillas simples.....	12
Comillas dobles.....	12
Heredoc.....	12
Nowdoc.....	12
Expresiones regulares en PHP.....	13
Operadores.....	13
Funciones.....	17
Paso de parámetros.....	17
Valor devuelto por la función.....	19
Versión 7 de PHP.....	20
Alcance de las variables.....	21
Estructuras de control.....	21
Localización y diseño del código.....	22
Control del buffer de salida – <i>Output Buffering</i> –.....	23
Instrucciones básicas.....	24
Inclusión de código.....	24
Envío de cabeceras HTTP.....	24
Finalización de un script.....	24
Funciones de cadenas útiles en la web.....	24
Arrays. Arrays superglobales.....	25
Arrays superglobales.....	28
Recuperación de los parámetros de un script.....	29
Seguridad – filtrado de datos.....	30
Manejo de archivos.....	31
Archivos de texto.....	31
Subir archivos al servidor.....	31
Archivos JSON.....	32
Configuración Apache y PHP.....	34
Directivas del servidor Apache: httpd.conf.....	34
Directivas de PHP: php.ini.....	35
Orientación a objetos.....	37

Definición de una clase.....	37
Creación de objetos.....	37
Asignación y Comparación de objetos.....	38
Herencia.....	38
Constructor y destructor.....	38
Propiedades.....	39
Propiedades y métodos estáticos.....	39
Acceso a las propiedades.....	39
Constantes.....	39
Clases y métodos abstractos.....	40
Clases y métodos finales.....	41
Clases anónimas.....	41
Interfaces.....	41
Traits.....	42
Sobrecarga.....	42
Métodos mágicos.....	43
Patrón Singleton.....	46
Acceso a BD MySQL.....	47
Introducción.....	47
La extensión PDO.....	47
La extensión mysqli.....	49
Conectar con la base de datos.....	52
Ejecutar SQL.....	54
Recuperar datos.....	54
Parametrizar el SQL.....	55
Transacciones.....	58
Cerrar la conexión.....	59
Mantenimiento del Estado HTTP – Cookies y Sesiones.....	60
Introducción.....	60
Cookies.....	60
Sesiones.....	62
Configuración PHP.INI de la sesión.....	63
Funciones de gestión de la sesión.....	64
Protocolo HTTP.....	66
Mensajes HTTP.....	66
Peticiones HTTP (Request).....	68
Respuestas HTTP (Response).....	71
Servicios web.....	76
SOAP (Simple Object Access Protocol).....	76
REST (Representational State Transfer).....	78
Aspectos de seguridad.....	84
Mensajes de error.....	84
Register Globals (obsoleta/eliminada).....	84
Magic Quotes (obsoleta/eliminada).....	84
Re-autenticaciones.....	84
Hash de contraseñas seguro.....	85

Plantillas.....	87
Frameworks PHP:.....	88
Lenguaje Javascript.....	89
Localización del código Javascript.....	90
Expresiones regulares en Javascript.....	90
Acceso a los elementos de la página.....	91
Acceso a las propiedades de elementos de la página.....	92
El modelo de objetos.....	92
Validación de formularios en el cliente.....	93
Capturar los eventos necesarios.....	94
Detección de los errores.....	95
Informar al usuario.....	97
Ejemplos sencillos.....	98
Librerías externas Javascript.....	100
jQuery.....	100
DataTables: plugin de tablas para jQuery.....	100
Gráficos: Chart.js y CanvasJS.....	100
AJAX.....	101
Anexo – MySQL desde phpMyAdmin.....	102
Introducción.....	102
Creación de bases de datos.....	102
Tablas. Motores. Tipos de datos.....	102
Gestión de usuarios.....	103
Relaciones.....	104
Exportación e Importación.....	105

Introducción

El siguiente manual no es un material exhaustivo de todos los elementos que conforman la generación dinámica de sitios web sino un estudio de los conceptos más importantes, así como el establecimiento de la comunicación entre todos ellos: básicamente es tener claro dónde tiene que estar cada elemento.

Aunque HTML y CSS forma parte de la materia de módulos del primer curso, conviene profundizar más en ciertos aspectos: en concreto en los formularios HTML para su posterior interacción con lenguajes de script en el cliente y en el servidor, y sus diversos mecanismos de comunicación.

Con respecto al código HTML generado, no siempre generaremos XHTML, que quizás sería lo adecuado. El WWC ha elegido el HTML5 en vez del XHTML2, puesto que es más sencillo de usar y sobre todo menos estricto de validar. Además si queremos indicar que nuestra página está diseñada para HTML5 basta con indicar como primera línea del documento: `<!DOCTYPE html>`, en vez de todas las variedades en las cabeceras utilizadas anteriormente (strict transactional frameset etc...). No hay que indicar ninguna referencia a ningún DTD, puesto que no se basa en SGML.

También es importante conocer CSS y al menos alguna de las librerías existentes con plantillas, clases, etc. para poder 'vestir' una web de una manera sencilla, rápida, cumpliendo estándares de diseño y que adapte su contenido de una manera adecuada a los distintos tamaños de dispositivos utilizados para acceder a la web. Twitter Bootstrap podría ser una de dichas librerías.

Instalación del entorno y herramientas de programación

Para el desarrollo de aplicaciones web necesitamos, al menos:

- Un servidor web (Apache, IIS, ...)
- Un lenguaje de programación para ejecutar código en el servidor y generar dinámicamente la página solicitada. Actualmente, casi cualquier lenguaje puede realizar esta tarea: PHP, C# y VB.NET (ASPX) , Java (JSP, Servlets), PERL, Python, etc.

Incluso un lenguaje como JavaScript (ECMAScript), típico lenguaje de ejecución en el lado del cliente (~navegador,) puede ejecutarse en el lado del servidor utilizando una alternativa al Apache como es Node.js Esta plataforma es muy adecuada para webs que reciben muchas peticiones, pues evita que lenguajes como Java/PHP creen un hilo para cada petición web que podría colapsar fácilmente el servidor por la memoria (~2Mb/hilo, 8Gb RAM ~4000 peticiones) y cambios de contexto necesarios.

- Un gestor de base de datos: Oracle, PostgreSQL, DB2, SQL Server, MySql (o MariaDb), etc.

La instalación del trío servidor web / lenguaje de programación / gestor de base de datos puede hacerse individualmente (configurando adecuadamente sus módulos y conexiones), o se pueden instalar paquetes tipo XAMPP (LAMP | WAMP | MAMP). La instalación de este tipo de software nos deja el sistema preparado para empezar el desarrollo de la aplicación web (no para producción).

Opcionalmente podremos utilizar distintos frameworks – como Symfony o Laravel – que nos ayudarán a programar utilizando conocidos patrones (MVC) que mantendrán nuestro código ordenado y fácilmente extensible.

Como herramientas que nos ayudan a la codificación tendremos:

- Simples editores de texto que, al menos, reconozcan la sintaxis PHP y sus librerías estándar (Notepad++, Sublime Text, Atom, ...)
- IDEs como NetBeans + plugin PHP:

Tools > Options > General > Proxy Settings (en el caso de ser necesario)

Tools > Plugins > Available Plugins > Search:PHP > Elegir PHP (por ejemplo) > Install > Reiniciar IDE

File > New Project > PHP

Formularios HTML

<FORM ACTION='URL destino de los datos del formulario' METHOD='GET | POST'>

Definición de los controles del formulario

</FORM>

En el atributo ACTION ponemos la URL destino de los datos especificados en el formulario. La URL puede ser una aplicación CGI, una aplicación PHP, un servlet de Java, una dirección de correo (con el formato `mailto:usuario@servidor`), etc.

Método de envío: GET o POST

En el atributo METHOD pondremos “GET” (por defecto) o “POST” para especificar el método utilizado para enviar por HTTP los datos del formulario.

– El método GET envía los datos como parte de la URL, de la siguiente forma:

`URL?control1=dato+del+control+1&control2=dato2&control3=dato3&...`

Debido a que la longitud de la dirección del navegador es limitada, GET está recomendado para formularios pequeños. Además, como los datos van en la propia URL, nunca se deberían enviar contraseñas o datos sensibles utilizando este método. Además estamos limitados a enviar caracteres de texto.

Una gran ventaja del envío por GET es que nos permite parametrizar una página en la propia URL, es decir, podemos enviar o utilizar enlaces donde ya van los datos que necesita.

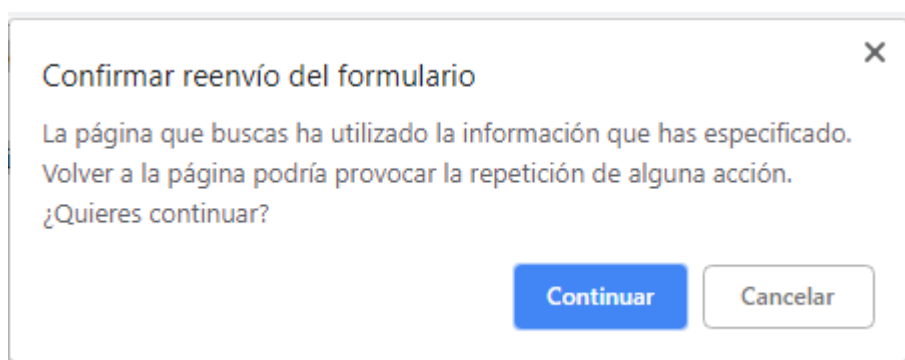
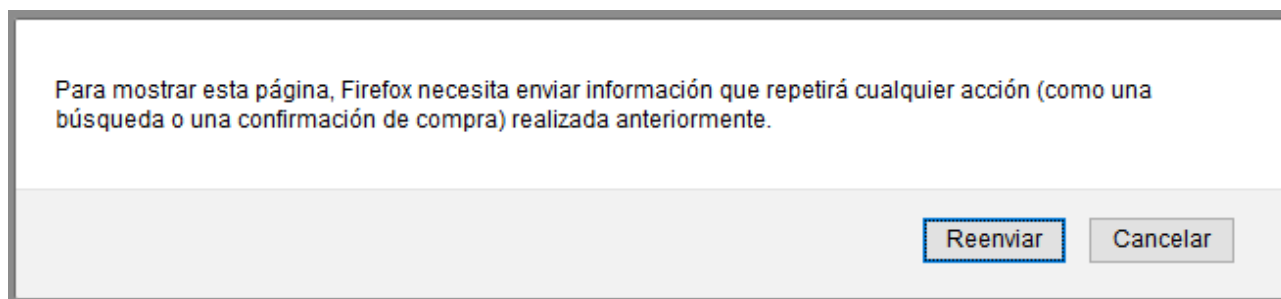
Por otra parte, la especificación recomienda utilizar el método GET únicamente cuando el procesamiento del formulario sea idempotente, que el envío de los datos no produzca efectos en el servidor (solo en la pantalla del cliente). Es decir, este método es perfecto para realizar consultas, pero si existen cambios en la base de datos se debería usar POST.

En todo caso, también se considera idempotente si los cambios en el servidor de varias peticiones es el mismo que si sólo hubiera habido una.

– El método POST envía los datos del formulario dentro del cuerpo de la petición HTTP, de manera oculta al usuario. En general usaremos POST cuando:

- se vayan a producir cambios en la base de datos del servidor
- los datos contengan caracteres no ASCII
- haya muchos datos (la URL tiene un límite físico)
- queremos ocultar un campo (tipo *hidden*), pues con GET se vería en la URL (aunque los campos ocultos no lo están realmente, pues se puede ver el código fuente del formulario)
- por supuesto, cuando enviemos una contraseña

Como se puede ver en las imágenes (del FireFox y Chrome respectivamente), el navegador nos avisa cuando se va a reenviar la misma información que se envió anteriormente (con sus posibles efectos colaterales). Esto suele ser producido por una actualización (F5) de la página.



Controles del formulario

Caja de texto

```
<INPUT TYPE='text' NAME='nombre' ID='id' SIZE='tamaño' VALUE='valorPorDefecto'>
```

Caja de texto para contraseñas (visualiza asteriscos)

```
<INPUT TYPE='password' NAME='nombre' ID='id' SIZE='tamaño' VALUE='valorPorDefecto'>
```

Conjunto de opciones exclusivas

```
<INPUT TYPE='radio' NAME='nombreDelGrupo' VALUE='1'>Uno
```

```
<INPUT TYPE='radio' NAME='nombreDelGrupo' VALUE='2'>Dos
```

```
<INPUT TYPE='radio' NAME='nombreDelGrupo' VALUE='3' CHECKED='CHECKED'>Tres
```

Casilla de verificación

```
<INPUT TYPE='checkbox' NAME='nombre' VALUE='valor'>
```

Control oculto: (usado para enviar información oculta al servidor)

```
<INPUT TYPE='hidden' NAME='nombre' ID='id' VALUE='valor'>
```


Control para seleccionar un archivo

```
<INPUT TYPE='file' NAME='nombre' ID='id'>
```

Caja de texto multilinea

```
<TEXTAREA NAME='nombre' ID='id' ROWS='nºFilas' COLS='nºColumnas'>
    texto de inicialización de la caja de texto
</TEXTAREA>
```

Control lista

```
<SELECT NAME='nombre' MULTIPLE='MULTIPLE' SIZE='nºopcionesVisibles'>
    <OPTION VALUE='valorOpción1'>Texto opción 1</OPTION>
    <OPTION VALUE='valorOpción2' SELECTED>Texto opción 2</OPTION>
    <OPTION VALUE='valorOpción3'>Texto opción 3</OPTION>
</SELECT>
```

En el caso de que sea una lista múltiple, el name tendrá la forma NAME='nombre[]', dando a entender que cuando se recoja el resultado será un array para poder albergar varias opciones.

Botón de envío de datos

```
<INPUT TYPE='submit' VALUE='Enviar'>
```

Botón de reseteo de datos

```
<INPUT TYPE='reset' VALUE='Borrar'>
```

Botón sin comportamiento predeterminado (se capturará el evento onClick)

```
<INPUT TYPE='button' VALUE='Texto del botón' onClick='código Javascript...'>
```

Existen multitud de nuevos tipos de controles introducidos con HTML5. Entre ellos estarían *number, range, date, datetime, month, week, color, email, url, etc.* Aún así, tenga en cuenta los problemas de compatibilidad entre navegadores. Es posible que el comportamiento no sea el mismo en todos ellos, o que directamente alguno no funcione.

Al enviar (submit) el formulario, se enviarán al servidor los datos de aquellos controles que tengan un NAME asignado, teniendo en cuenta las siguientes excepciones:

- los grupos (RADIO) no seleccionados
- las casillas de verificación (CHECKBOX) no seleccionadas
- las listas (SELECT) sin ningún elemento seleccionado (sólo puede ocurrir en aquellas con el atributo SIZE>1, es decir, con más de un elemento visible)

En el caso de querer enviar múltiples datos similares, de nuevo se utilizará la sintaxis NAME='nombre[]' en todos los controles equivalentes, de forma que se recogerá como un array en el lenguaje de servidor utilizado.

El valor de dicho NAME se utilizará como clave para acceder a ellos posteriormente en el lenguaje utilizado en el servidor para recuperarlos.

Lenguaje PHP

Lenguaje de script ejecutado en el servidor, cuyo objetivo principal es generar páginas web de manera dinámica. Es decir, la página se crea en el momento en que se solicita.

El código PHP se suele encontrar incrustado en el código HTML. De esta manera, las páginas resultantes serán la combinación del HTML/CSS/JS estático junto con el HTML/CSS/JS resultado del procesamiento de los distintos bloques PHP.

Los bloques de código PHP se encuentran entre las etiquetas `<?php ... ?>`. En el caso de que la directiva `short_open_tag true/false` esté activa, se puede usar `<? ... ?>`

El servidor web sabe distinguir cuándo le piden un archivo con extensión htm/html de un php. En el primer caso, el archivo se sirve tal cual. En el segundo caso, el servidor web le da a al intérprete PHP cada bloque para que lo ejecute y recupera el resultado generado dinámicamente que se combina con la parte estática para devolver una página que, obviamente, ya no contiene ningún bloque PHP que, por otra parte, el navegador no entendería.

Dado que los servidores web son altamente configurables, algunos administradores lo configuran de tal manera que aunque la extensión sea htm/html, se pase al intérprete PHP de tal manera que el cliente no puede saber por la URL qué lenguaje de script se está ejecutando en el servidor.

Toda la información que necesitemos saber sobre versión y configuración de PHP la obtendremos llamando a la función `phpinfo()`;

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

La función anterior nos permite visualizar mucha información en la pantalla, pero si lo que nos interesa es conocer desde el código la versión actual de PHP podemos utilizar la función `phpversion()`. A veces también puede ser útil la función `function_exists("nombreDeLaFunción")`.

Un artículo interesante que nos da una visión general del PHP como lenguaje de programación se puede visualizar en <https://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/> o una alternativa en el documento “PHP: Alternativa de un mal diseño”

Variables

Las variables siempre comienzan con un carácter \$ y son *case sensitive*.

Las constantes se crean utilizando **define('nombreConstante', valor);** y posteriormente se utiliza directamente el nombre de la constante, que no lleva \$ en ningún momento. A partir de PHP 5.3 se puede usar la (más moderna) instrucción **const NOMBRE_CONSTANTE = valor;**

Los tipos se asignan dinámicamente en función del valor asignado. Una misma podría tener varios tipos de datos distintos a lo largo de su vida, aunque no parece una práctica muy recomendable abusar de esta característica.

Para saber si una variable está definida se utiliza **isset(\$var1[, \$var2[, \$var3...]])**, que devolverá un *true* si la variable está definida **y no es NULL**. Se le puede pasar un n.º variable de parámetros, y los evaluará ordenadamente finalizando si encuentra alguno no definido. Como parámetro se le puede pasar también una entrada en un array, pero hay que tener en cuenta que devolverá *false* si existe y es nula.

Generación de HTML

Para generar HTML desde PHP habitualmente se utiliza **echo** o **print**, a las que le pasamos la cadena que queremos generar en HTML. La diferencia entre ambas es que **print** devuelve 1, para ser utilizada en expresiones.

También existe **printf**, a la que le podemos pasar una cadena con los caracteres de formato adecuados anteceditos por un %, y tantos parámetros a continuación como caracteres de formato hayamos indicado. Los parámetros de formato pueden ser b(binario), c(carácter del nº indicado), d(entero), e(notación científica), f(nº real con punto decimal), o(octal), s(cadena), x(hexadecimal en minúsculas), X(hexadecimal en mayúsculas), etc. Entre el % se le puede especificar el nº de decimales que nos interesa visualizar (%.2f para visualizar dos decimales, por ejemplo), y también se puede especificar que la cadena resultante tenga un cierto número de caracteres, rellenando con espacios en blanco, o con cualquier otro símbolo, y justificado a la izquierda o a la derecha.

La función **sprintf** también es muy utilizada porque tiene la misma funcionalidad que **printf** pero no visualiza (o genera HTML) inmediatamente, sino que devuelve la cadena con formato generada para utilizarla desde el código como nos sea más conveniente.

Evaluación de cadenas

Las [cadenas](#) en PHP se pueden especificar de cuatro maneras distintas:

- entre comillas simples
- entre comillas dobles
- sintaxis *heredoc*
- sintaxis *nowdoc* (>PHP 5.3.0)

Comillas simples

Todos los caracteres especificados encerrados entre comillas simples se evaluarán como dichos caracteres, excepto `\'` para una comilla simple y `\\` para el propio carácter de *backslash*.

Comillas dobles

En las cadenas encerradas entre comillas dobles se evaluarán los caracteres de escape habituales (`\n`, `\t`, `\\`, `\'`, etc.). Pero lo más importante es que las variables se expanden con su valor, pudiendo utilizar sintaxis sencilla y compleja. En la sencilla, el parser busca el signo `$` y la mayor cantidad de caracteres posibles para identificar el nombre de la variable. También funciona con arrays numéricos y con array asociativos pero sin entrecomillar la clave. La sintaxis compleja la usamos para expresiones más complicadas, como accesos a arrays o propiedades de objetos: básicamente consiste en encerrar entre llaves la expresión que utilizaríamos directamente fuera de la cadena.

Heredoc

Se utiliza el operador `<<< IDENTIFICADOR` y a continuación comienza una línea donde va el string ocupando las líneas que sean necesarias. Para acabar el string se pondrá únicamente, y en una línea nueva, el IDENTIFICADOR seguido de punto y coma (;)

El texto heredoc se comporta como las comillas dobles (en cuando a evaluación de variables), pero sin dichas comillas. De hecho, se pueden utilizar comillas tranquilamente.

```
<?php
echo <<<FIN
<TR><TD>$datoColumna1</TD><TD>$datoColumna2</TD>
...
</TABLE>
El total es $total
FIN;
?>
```

Nowdoc

Nowdoc es a los string con comillas simples lo mismo que Heredoc lo es a los string con comillas dobles, es decir, no evalúa nada. Su sintaxis es igual a Heredoc excepto a que el IDENTIFICADOR se especifica entre comillas simples en su apertura.

```
<?php
echo <<<'ENDOFTEXT'
...
ENDOFTEXT;
?>
```

Es perfecto para visualizar código PHP o mucho texto que no queremos que se analice. Es similar a la construcción `<![CDATA[]]>` de SGML, donde se declara un bloque de texto que no se analiza.

Expresiones regulares en PHP

```
int preg_match (string $patrón,string $texto [,array &$matches [,int $flags=0 [,int $offset=0 ]]])
```

Busca encajar el texto con el patrón especificado.

Devuelve 1 si encaja, 0 si no, y FALSE si ocurre algún error.

Si se especifica matches se almacenarán los resultados de la evaluación. *matches[0]* contendrá el texto que encajó con todo el patrón y los sucesivos índices el valor de los grupos (subpatrones entre paréntesis) de la expresión regular.

Operadores

A continuación mostramos una tabla con la **precedencia** y la **asociatividad** de los operadores de PHP, lo cual determina cómo se agrupan las expresiones.

Asociativo	Operadores	Info adicional
N/A	clone new	clone and new
IZQ	[array()
DER	**	aritmética
DER	++ -- ~ @	tipos e inc/dec
N/A	instanceof	tipos
DER	!	lógico
IZQ	* / %	aritmética
IZQ	+ - .	aritmética y string
IZQ	<< >>	bit a bit
N/A	< <= > >=	comparación
N/A	== != === !== <> <=>	comparación
IZQ	&	bit a bit y referencias
IZQ	^	bit a bit
IZQ		bit a bit
IZQ	&&	lógico
IZQ		lógico
DER	??	comparación
IZQ	? :	ternario
DER	= += -= *= **= /= .= %= &= = = <=> >>=	asignación
IZQ	and	lógico
IZQ	xor	lógico
IZQ	or	lógico

Operadores aritméticos

Example	Name	Result
<code>-\$a</code>	Negation	Opposite of <i>\$a</i> .
<code>\$a + \$b</code>	Addition	Sum of <i>\$a</i> and <i>\$b</i> .
<code>\$a - \$b</code>	Subtraction	Difference of <i>\$a</i> and <i>\$b</i> .
<code>\$a * \$b</code>	Multiplication	Product of <i>\$a</i> and <i>\$b</i> .
<code>\$a / \$b</code>	Division	Quotient of <i>\$a</i> and <i>\$b</i> .
<code>\$a % \$b</code>	Modulus	Remainder of <i>\$a</i> divided by <i>\$b</i> .

Operadores de bits

Example	Name	Result
<code>\$a & \$b</code>	And	Bits that are set in both <i>\$a</i> and <i>\$b</i> are set.
<code>\$a \$b</code>	Or (inclusive or)	Bits that are set in either <i>\$a</i> or <i>\$b</i> are set.
<code>\$a ^ \$b</code>	Xor (exclusive or)	Bits that are set in <i>\$a</i> or <i>\$b</i> but not both are set.
<code>~ \$a</code>	Not	Bits that are set in <i>\$a</i> are not set, and vice versa.
<code>\$a << \$b</code>	Shift left	Shift the bits of <i>\$a</i> <i>\$b</i> steps to the left (each step means "multiply by two")
<code>\$a >> \$b</code>	Shift right	Shift the bits of <i>\$a</i> <i>\$b</i> steps to the right (each step means "divide by two")

Operadores de comparación

Example	Name	Result
<code>\$a == \$b</code>	Equal	TRUE if <i>\$a</i> is equal to <i>\$b</i> .
<code>\$a === \$b</code>	Identical	TRUE if <i>\$a</i> is equal to <i>\$b</i> , and they are of the same type. (introduced in PHP 4)
<code>\$a != \$b</code>	Not equal	TRUE if <i>\$a</i> is not equal to <i>\$b</i> .
<code>\$a <> \$b</code>	Not equal	TRUE if <i>\$a</i> is not equal to <i>\$b</i> .
<code>\$a !== \$b</code>	Not identical	TRUE if <i>\$a</i> is not equal to <i>\$b</i> , or they are not of the same type. (introduced in PHP 4)
<code>\$a < \$b</code>	Less than	TRUE if <i>\$a</i> is strictly less than <i>\$b</i> .
<code>\$a > \$b</code>	Greater than	TRUE if <i>\$a</i> is strictly greater than <i>\$b</i> .
<code>\$a <= \$b</code>	Less than or equal to	TRUE if <i>\$a</i> is less than or equal to <i>\$b</i> .
<code>\$a >= \$b</code>	Greater than or equal to	TRUE if <i>\$a</i> is greater than or equal to <i>\$b</i> .

Operadores de incremento/decremento

Example	Name	Effect
<code>++\$a</code>	Pre-increment	Increments <i>\$a</i> by one, then returns <i>\$a</i> .
<code>\$a++</code>	Post-increment	Returns <i>\$a</i> , then increments <i>\$a</i> by one.
<code>--\$a</code>	Pre-decrement	Decrements <i>\$a</i> by one, then returns <i>\$a</i> .
<code>\$a--</code>	Post-decrement	Returns <i>\$a</i> , then decrements <i>\$a</i> by one.

Operadores lógicos

Example	Name	Result
<code>\$a and \$b</code>	And	TRUE if both <i>\$a</i> and <i>\$b</i> are TRUE .
<code>\$a or \$b</code>	Or	TRUE if either <i>\$a</i> or <i>\$b</i> is TRUE .
<code>\$a xor \$b</code>	Xor	TRUE if either <i>\$a</i> or <i>\$b</i> is TRUE , but not both.
<code>! \$a</code>	Not	TRUE if <i>\$a</i> is not TRUE .
<code>\$a && \$b</code>	And	TRUE if both <i>\$a</i> and <i>\$b</i> are TRUE .
<code>\$a \$b</code>	Or	TRUE if either <i>\$a</i> or <i>\$b</i> is TRUE .

Operadores de array

Example	Name	Result
<code>\$a + \$b</code>	Union	Union of <i>\$a</i> and <i>\$b</i> .
<code>\$a == \$b</code>	Equality	TRUE if <i>\$a</i> and <i>\$b</i> have the same key/value pairs.
<code>\$a === \$b</code>	Identity	TRUE if <i>\$a</i> and <i>\$b</i> have the same key/value pairs in the same order and of the same types.
<code>\$a != \$b</code>	Inequality	TRUE if <i>\$a</i> is not equal to <i>\$b</i> .
<code>\$a <> \$b</code>	Inequality	TRUE if <i>\$a</i> is not equal to <i>\$b</i> .
<code>\$a !== \$b</code>	Non-identity	TRUE if <i>\$a</i> is not identical to <i>\$b</i> .

Notas sobre el comportamiento de algunos operadores

- El operador @ pertenece al control de errores, pues antenponiendo @ a una instrucción, se ignoran sus mensajes de error.
- También se puede considerar un operador de ejecución los *backticks*, metiendo una cadena entre `apóstrofes invertidos`, pues se ejecutará como un comando del shell y devolverá su salida textual (es análogo a una llamada a *shell_exec(string \$comando)*)
- La diferencia entre el operador == y el operador === es que el primero compara si sus operandos son iguales, y el segundo compara también que sus tipos sean iguales.

```
<?php
echo "1"==true?'Cierto':'Falso'; // Cierto
echo 1==true?'Cierto':'Falso'; // Cierto
echo 0==false?'Cierto':'Falso'; // Cierto
echo "1"===true?'Cierto':'Falso'; // Falso
echo 1===true?'Cierto':'Falso'; // Falso
echo 0===false?'Cierto':'Falso'; // Falso
echo 0===null?'Cierto':'Falso'; // Cierto
echo 0===null?'Cierto':'Falso'; // Falso
echo false===null?'Cierto':'Falso'; // Cierto
echo false===null?'Cierto':'Falso'; // Falso
echo "777"===777?'Cierto':'Falso'; // Cierto
echo "777"===777?'Cierto':'Falso'; // Falso
?>
```

- La razón para tener las dos variaciones diferentes del mismo operador (and y &&, or y ||) es que operan con precedencias diferentes (and y or tienen la mínima). Por cierto, ambos operadores están cortocircuitados (no evalúan el segundo operador si no es necesario).

```
$falso = true && false;
echo $falso?'Cierto':'Falso';

$deberiaSerFalso = true and false;
echo $deberiaSerFalso?'Cierto':'Falso';

$cierto = false || true;
echo $cierto?'Cierto':'Falso';

$deberiaSerCierto = false or true;
echo $deberiaSerCierto?'Cierto':'Falso';

/*
Posiblemente se espere la siguiente salida: FalsoFalsoCiertoCierto
Pero en realidad se obtiene: FalsoCiertoCiertoFalso

Esto es debido a que la asignación tiene menos prioridad que && y || pero más que and y or
y se realiza antes la asignación de la variable al valor que tiene a la derecha del operador =
*/
```

- El operador ternario merece un comentario por tener una implementación diferente a cualquier otro lenguaje de programación conocido, pues es asociativo a la izquierda !!!

```
$dato=false; echo "Dato: " . $dato?'T':'F'; // qué visualizará?

echo true ? 0 : true ? 77 : 99;
// Visualiza 99 (true ? 0 : true) ? 77 : 99

echo true ? 0 : (true ? 77 : 99);
// Aquí sí visualiza 0, posiblemente lo que esperábamos

echo true ? 1 : true ? 77 : 99;
// Visualiza 77 (true ? 1 : true) ? 77 : 99
```

Observando el comportamiento de estos últimos operadores, el consejo es obvio: hay que usar paréntesis incluso cuando parezcan innecesarios para prevenir comportamientos inesperados.

- Si en el operador ternario (a partir de PHP 5.3) se deja en blanco el segundo operando, se convierte en un operador binario denominado *Elvis operator*:

```
$resultado = $dato ?: 'valor si se evalúa como falso';
```



La variable \$resultado será 'valor si se evalúa como falso' cuando \$dato sea 0, '0', false o null. En el caso de que \$dato no esté asignada, entonces se producirá un error. Este comportamiento (la necesidad de que la variable esté previamente asignada) hace que la utilidad de este operador sea menor que la del operador ?? que veremos a continuación

- Operadores nuevos en PHP 7 son:
 - ?? Operador fusión de null: devuelve el primer operando de izquierda a derecha que exista y no sea NULL.

```
$d='algo';
echo $a??$b??$c??$d??$z;
// Visualiza algo

$d=NULL;
echo $a??$b??$c??$d??$z;
// Visualiza Notice: Undefined variable: z ...
```

Por ello la línea (siempre recordando que es a partir de PHP 7.0):

```
$dato=$_POST['dato']??'dato por defecto';
```

puede sustituir a la utilizadísima:

```
$dato=isset($_POST['dato'])?$_POST['dato']:'dato por defecto';
```

- <=> Operador nave espacial: devuelve un valor negativo, cero o positivo en función de si el primer operando es menor, igual o mayor que el segundo, respectivamente.

Funciones

```
<?php
    function nombreDeLaFuncion($parametro1, $parametro2, ... ) {
        ... código ...
        return $valorDevuelto;
    }
?>
```

- El nombre de las funciones **no son** sensibles a mayúsculas y minúsculas.
- Se pueden proporcionar valores por defecto a los últimos parámetros, asignándole el valor en su declaración
- PHP no admite la sobrecarga de funciones
- Para devolver el valor de la función se usa *return*, finalizando su ejecución inmediatamente
- Dentro de la función puede ir cualquier código válido, incluso funciones y clases !

```
<?php
    function f($a,$b) {
        function suma($s1,$s2) { return $s1+$s2; }
        echo suma($a,$b);
    }
    f(5,7);

    echo suma(9,9);
    // Llama directamente a una función interna y funciona. Uf!
?>
```

Este comportamiento es debido a que todas las funciones y clases de PHP tienen ámbito global. Se pueden llamar desde fuera de una función incluso si fueron definidas dentro, y viceversa.

Paso de parámetros

Los parámetros se pasan a las funciones a través de una lista de nombres delimitada por comas. PHP permite:

- el paso por valor (por defecto): el parámetro se inicializa con el valor pasado a la función.
- el paso por referencia: el parámetro apunta a la variable pasada a la función, por lo que cambiar el parámetro equivale a cambiar el valor de la variable. Se crea anteponiendo un ampersand (&) al nombre del parámetro en la definición de la función.

```
function despedida(&$mensaje) { $mensaje .= '<br>Hasta luego y gracias !'; }
$saludo='Hola a todos';
despedida($saludo);
echo $saludo;
```

Hola a todos
Hasta luego y gracias !

En versiones antiguas de PHP se permitía decidir si el paso se realizaba por valor o por referencia (incluyendo también un & antes del nombre del parámetro al llamar a la función). A partir de la v.5.3.0 se emitía un warning ("call-time pass-by-reference") y a partir de la v.5.4.0 directamente no está permitido y salta un error.

La utilización de referencias se puede utilizar en cualquier momento, no únicamente en el paso de parámetros a una función:

```
$nombre='Jose'; $alias=&$nombre; $alias='Pepe'; echo $nombre; // visualiza Pepe
```

- parámetros por defecto: se crean igualando a una expresión constante el valor del parámetro en la definición de la función. Los valores por defecto no pueden tener a su derecha un parámetro sin valor por defecto (esto se hace así para que no haya equívocos a la hora de asignarlos).

```
function separa ($dato1,$dato2,$separador='<br>') {
    return "$dato1$separador$dato2";
}

echo separa('Hola','Amigos','<hr>');
echo separa('Hasta','Luego');
```

- parámetros de longitud variable: se pone ... antes del parámetro, que se convertirá automáticamente en un array con tantos datos como parámetros se pasen a la función. Tiene que ser el último parámetro de la función. El token ... también puede servir para convertir un array en una lista de parámetros al llamar a una función.

```
function variosParametros (...$datos) {
    foreach($datos as $dato)          //$datos es un array
        echo $dato.'<br>';
}

variosParametros(1,2,3,4,5,"siete",8);

function suma($a,$b,$c) {
    return $a+$b+$c;
}

$sumandos=array(1,2,3);
echo "La suma es " . suma(...$sumandos);
echo " y 4+2+5 son " . suma(...[4,2,5]);
```

En el paso de parámetros en PHP, tradicionalmente, no se podía especificar su tipo de dato. A partir de PHP 5 se incluyeron los *type hints* entre los que se encontraban array, object, mixed (varios tipos pero no necesariamente todos), callable, el nombre de una clase o de un interface.

A partir de PHP 7 ya se pueden especificar los tipos bool, float,int,string e iterable.

El tipo iterable (introducido en PHP 7.1), acepta cualquier array u objeto que implemente el **interfaz Traversable**. Este interfaz sirve para permitir que una clase pueda recorrerse con un foreach. Por cierto, el interfaz Traversable es abstracto y sin métodos, pues sirve de base para otros interfaces como Iterator, que obliga a implementar los métodos current, key, next, rewind, y valid.

Valor devuelto por la función

Los valores son devueltos usando la sentencia opcional `return`. Se puede devolver cualquier tipo, incluidos arrays y objetos. Esto causa que la función finalice su ejecución inmediatamente y pase el control de nuevo a la línea desde la que fue llamada.

Una función no puede devolver múltiples valores, pero se pueden obtener resultados similares devolviendo un array.

```
function cincoPrimos() { return array (2, 3, 5, 7, 11); }  
list ($primero,$segundo,$tercero,$cuarto,$quinto) = cincoPrimos();  
echo "Los 5 primeros primos son $primero,$segundo,$tercero,$cuarto y $quinto";
```

PHP 7 permite declarar en la función el tipo de dato a devolver (similar a las declaraciones de tipo de los argumentos, vista anteriormente). Para realizarlo, debemos especificar `:tipoDevuelto` después de la declaración de la función y antes de abrir la llave para especificar el código:

```
function potencias(float $numero, int $cuantas) : array {  
    $resultados=[];  
    $calculo=1;  
    for($i=0;$i<$cuantas;$i++) {  
        $resultados[]=$calculo;  
        $calculo*=$numero;  
    }  
    return $resultados;  
}  
  
print_r(potencias(3.5,7));  
  
Array ( [0] => 1 [1] => 3.5 [2] => 12.25 [3] => 42.875 [4] => 150.0625 [5] => 525.21875 [6] => 1838.265625 )
```

Versión 7 de PHP

A finales de 2015 salió la versión 7. La versión 6 se abandonó debido a problemas de tratamiento de las cadenas Unicode, y sus mejoras fueron incorporadas en las últimas versiones 5.X

Entre las mejoras incluidas en la versión 7 se encuentran:

- Gran mejora en el rendimiento, lo cual lo pone en mejor posición para rivalizar con opciones como HHVM (Hip Hop Virtual Machine), la máquina virtual de Facebook que compila y ejecuta PHP
- Uso de *jsond* un nuevo parser JSON debido a problemas de licencia con el anterior
- Nuevos operadores:
 - spaceship (nave espacial) `<=>`
 - null coalescing (fusión de null) `??`
- Nuevas declaraciones de tipos en los argumentos de funciones y en sus valores de retorno

Debido especialmente a esta última característica de poder definir los tipos de argumentos y valores de retorno, es interesante comentar la existencia del constructor *declare*, que permite establecer directivas de ejecución para un bloque determinado de código afectado por la directiva o, si no se especifica, afecta a todo el código que le sigue

```
declare(directiva=valor) { códigoAfectado }      o      declare(directiva=valor);
```

Además de las directivas *tick* y *encoding*, PHP 7 añadió la directiva ***strict_types*** que permite indicar si queremos que, en el caso de que se encuentre un parámetro de función o valor de retorno de un tipo distinto al especificado, se intente *castear* (con un valor 0) o bien se genere directamente un error (con un valor 1).

```
declare(strict_types=1);
function sum(int $a, int $b):int { return $a + $b; }
echo sum(8.9,3); // error, no coinciden los tipos float e int
```

```
declare(strict_types=0); // Igualar a 0 es como no declarar la directiva ...
function sum(int $a, int $b):int { return $a + $b; }
echo sum(8.9,3); // Visualiza 11, no da error, convierte 8.9 en 8 ...
```

```
declare(strict_types=1);
function sum(int $a, int $b):int { return $a + $b; }
echo sum("11",4); // Error con tipos estrictos, funciona en caso contrario
```

La única excepción es si proporcionamos un entero y se espera un float, porque ahí no se pierde información, y la conversión es automática:

```
declare(strict_types=1);
function sum(float $a, float $b):float { return $a + $b; }
echo sum(4,3); // Visualiza 7, no da error
```

Alcance de las variables

- Locales .- accesibles únicamente en la sección del código en la que se han creado. Si es dentro de una función únicamente dentro de su código. Si es fuera de una función, en todo el código fuera de la función y de las clases.
- Globales .- accesibles en todas las partes del código. Para acceder, se antecede la variable con *global*
- Estáticas .- Se declaran con *static* dentro de una función, y sólo se accede dentro de la función pero conservan su valor entre distintas llamadas.

Estructuras de control

Son las estructuras de cualquier otro lenguaje de programación:

- Alternativas
 - *if / if – else / if – elseif*
 - *switch (expresión) { case: ... case: ... default: ... }*
- Repetitivas / Iterativas
 - *while (\$condicionBucle) / do – while (\$condicionBucle)*
 - *for (\$expresionInicial;\$condicionBucle;\$expresionFinalIteracion)*
 - *foreach (\$array as \$valor)*
 - *foreach (\$arrayAsociativo as \$clave=>\$valor)*
 - Instrucciones *break* y *continue*

Para incluir más de una instrucción dentro del ámbito de la estructura de control habitualmente se utilizan las llaves para encerrar dichas instrucciones (sintaxis C). Pero debido a la especial situación del código PHP, a menudo embebido en código HTML, la acumulación de llaves de cierre dificulta la rápida identificación de los bloques existentes.

Por ello (posiblemente) PHP permite encerrar las instrucciones afectadas abriendo el bloque con un carácter de dos puntos (:) y cerrando la estructura con las palabras clave *endif;* , *endswitch;* , *endwhile;* , *endfor;* o *endforeach;* , evitando el uso de las llaves {}.

Localización y diseño del código

Existen situaciones en que existe mucho HTML que, de estar dentro del PHP, habría que generar con `echo`'s. Por ello es posible incluirlo directamente abriendo y cerrando adecuadamente los bloques PHP pudiendo romper, aparentemente, las estructuras alternativas o repetitivas.

```
<?php const NUMERO = 7; ?>
<html>
<head>...</head>
<body>
    <h1>Tabla del <?php echo NUMERO; ?></h1>
    <table border='1'>
        <tr>
            <th>Operación</th>
            <th>Resultado</th>
        </tr>
        <?php
            for($i=1;$i<=10;$i++)
            {
                $resultado = NUMERO * $i;
            }
        ?>
        <tr>
            <td><?php echo NUMERO . " x $i"; ?></td>
            <td><?php echo $resultado; ?></td>
        </tr>
    <?php } ?>
    </table>
</body>
</html>
```

Podemos evitar un `echo` utilizando su forma corta: `<?= expresiónAVisualizar ?>` si `short_open_tag true` o a partir de la versión 5.4.0.

A continuación vemos una modificación del mismo código (con `<?=` y con `for: endfor;`)

```
<?php const NUMERO = 7; ?>
<html><body>
    <h1>Tabla del <?=NUMERO?></h1>
    <table border='1'>
        <tr>
            <th>Operación</th>
            <th>Resultado</th>
        </tr>
        <?php
            for($i=1;$i<=10;$i++):
                $resultado = NUMERO * $i;
            ?>
            <tr>
                <td><?=NUMERO . " x $i"?></td>
                <td><?=$resultado?></td>
            </tr>
        <?php endfor;?>
    </table>
</body></html>
```

Control del buffer de salida – *Output Buffering* –

Se denomina *Output Buffering* a la característica de PHP que nos permite no enviar al cliente la salida a medida que se va generando, sino que se almacena en un buffer controlado por el script.

Existen varias funciones PHP – `session_start`, `setcookie`, `header`, etc. – que no se pueden llamar si ya se ha producido alguna salida (estática o dinámica), pues modifican la cabecera HTTP de la respuesta del servidor y se supone que ésta ya se ha generado y ha viajado al cliente antes del primer carácter generado.

Tenga en cuenta que un simple carácter de espacio en blanco antes de un bloque `<?php ...` ya es una salida, o incluso aunque en el editor de texto no lo veamos, guardar el script en formato UTF-8 (en vez de UTF8-sin BOM – *Byte Order Mark* – indica el orden de los bytes, big-endian o little endian), también produce una salida y por consiguiente el conocido error:

Cannot send session cache limiter - headers already sent (output started at [script] on line ...)

Para evitar este error podemos programar adecuadamente el script almacenando internamente el contenido generado, comprobando las condiciones necesarias y llamando a las funciones en el orden correcto antes de generar dicho contenido.

O también podemos utilizar los mecanismos existentes en PHP para el control del buffer:

Directivas en los archivos de configuración

`output_buffering=Off|On|NumBytes`

Off deshabilita el buffer de salida

On la habilita de modo ilimitado (usar con cuidado)

Especifica el nº de bytes reservado para el buffer

`output_handler=función de PHP` (no de usuario) que recoge y gestiona el contenido generado

`implicit_flush=Off|On` Indica si envía inmediatamente al cliente cualquier salida generada

Funciones PHP de control del buffer

Podemos programar utilizando las funciones de control del buffer de salida (en cada página), para no depender del contenido del `php.ini` al que, probablemente, no tengamos acceso.

`ob_start()` Activa el almacenamiento en el buffer. Cualquier salida del script será almacenada en dicho buffer (excepto las cabeceras, que irán saliendo en su orden).

`ob_flush()`, `ob_end_flush()` Envía el contenido del buffer a la salida (y deshabilitándolo)

`ob_clean()`, `ob_end_clean()` Limpian el contenido del buffer (y deshabilitándolo)

Indicar que se pueden anidar buffers indicando `ob_start()` antes de cerrar el buffer previo.

Instrucciones básicas

Inclusión de código

Separar en archivos independientes puede ayudarnos a estructurar mejor nuestro código, así como no tener que repetirlo en distintos lugares. Para ello utilizaremos las instrucciones **include** y **require**, seguidas por la ruta al archivo cuyo contenido queremos añadir en esa posición.

La diferencia fundamental es como tratan la no existencia del archivo incluido. El *include* genera un *warning*, y el *require* genera un error y finaliza la ejecución del *script*.

Existen dos variantes (**include_once** y **require_once**) que evitan que un mismo archivo se incluya más de una vez.

Envío de cabeceras HTTP

Con **header(...)** podemos enviar directamente una cabecera HTTP (ver anexo).

```
header("HTTP/1.0 404 Not Found"); // Página no encontrada
header('Content-type: application/pdf'); // Vamos a generar un PDF
header("Location:URL"); // Redirección a otra página. Su uso más extendido
exit; // Si no queremos que siga ejecutando el script antes de irse ...
```

Como se envía información que viaja en la cabecera, no se puede enviar ningún contenido (estático o generado desde PHP) antes de la llamada a dicha instrucción, pues este envío ya habría generado su propia cabecera. Este comportamiento se puede evitar utilizando los mecanismos vistos anteriormente para habilitar el buffer de salida.

Finalización de un script

Podemos utilizar indistintamente **exit** o **die**. Son equivalentes. Son construcciones del lenguaje que finaliza el script en ese momento (ni ejecuta más código PHP ni da salida al HTML estático que quede por analizar). Se puede utilizar de 3 maneras:

- sin paréntesis
- pasándole entre paréntesis una cadena, que visualizará antes de finalizar
- pasándole entre paréntesis un entero: no se visualiza pero será el estado de salida (0==ok)

Funciones de cadenas útiles en la web

[urlencode / urldecode](#) Prepara una cadena para usar como URL sustituyendo los caracteres necesarios (espacios en blanco, etc.) por sus %códigos hexadecimales, y viceversa.

[nl2br](#) Convierte saltos de línea en saltos HTML (
)

[strip_tags](#) Elimina las etiquetas HTML de una cadena

Arrays. Arrays superglobales.

Los arrays en PHP son mapas ordenados. Un mapa es un tipo que asocia valores a claves.

- Las claves pueden ser enteros o cadenas. Cuando todas son enteros se suelen denominar arrays numéricos, y cuando todas son cadenas se suelen denominar arrays asociativos.
- Los valores de un array pueden ser de cualquier tipo, incluyendo otros arrays, hablando así de arrays de múltiples dimensiones.

La creación de un array se realiza con el constructor del lenguaje **array()** – o con **[]** a partir de PHP 5.4 – , especificando como argumentos una lista de elementos (clave=>valor) separada por comas (la última coma es opcional).

En el caso de especificar únicamente el valor (no incluyendo clave=>), se genera automáticamente una clave numérica cuyo valor será la clave numérica mayor existente + 1 (0 si no existe ninguna).

```
<?php
$coloresPiso = array('cocina'=>'Rojo', 'dormitorio'=>'Verde', 'salon'=>'Azul');
$colores = array('Rojo', 'Verde', 'Azul');
```

Para añadir un nuevo elemento a un array después de su creación, se usa directamente una asignación usando una clave nueva. Si la clave ya existía se cambia su valor (se actualiza), y si no se especifica una clave, se genera la clave numérica correspondiente (clave mayor + 1)

```
<?php
$coloresPiso['vestibulo']='Lila';
$colores[]='Lila';
```

En un mismo array pueden convivir sin problemas claves numéricas y claves de cadena.

El acceso a un elemento del array se realiza especificando el nombre de la clave entre corchetes.

Para saber si existe un elemento de un array se podría realizar de dos maneras:

- utilizando la función **array_key_exists** ([mixed](#) \$clave , array \$array) : bool
- utilizando la función **isset** ([mixed](#) \$var [, [mixed](#) \$. . .]) : bool teniendo en cuenta que devolverá FALSE si existe dicha clave pero su valor es NULL

Para recorrer automáticamente (en un bucle) los elementos del array podemos:

- Utilizar un bucle con una variable numérica para acceder a las claves numéricas. Un bucle for es el indicado, siempre teniendo en cuenta que la variable tome los valores correctos correspondientes al índice, pues en caso contrario obtendremos un error (fijarse que los índices podrían no ser correlativos). Así accederemos únicamente a los valores.
- Utilizar un bucle de tipo foreach (\$array as \$valor) o foreach(\$array as \$clave=>\$valor), que asigna automáticamente valores (y claves en la segunda forma)

```
for($i = 0; $i < count($colores); $i++)      foreach($colores as $color)
    echo $colores[$i] . '<br>';                echo "$color<br>";

foreach($coloresPiso as $hueco=>$color)      foreach($coloresPiso as $color)
    echo "$hueco de color $color<br>";        echo "$color<br>";
```

Para eliminar un elemento del array simplemente utilizamos la función **unset(\$variable)** pasándole el elemento del array que queremos eliminar. Tenga en cuenta que en el caso de eliminar una clave numérica, el resto de las claves no varían (no se reenumeran ni nada parecido)

Integrar los elementos de un array en una expresión de cadena

Como se puede ver en el siguiente ejemplo, hay que tener en cuenta el comportamiento de PHP para reconocer los valores de los elementos de un array dentro de una cadena (siempre “doble”).

Para forzar la evaluación se puede encerrar entre {} el valor de un array asociativo, aunque también se puede no entrecomillar la clave (uf!).

```
<?php
$nombre='Pepe';
$aficiones=array('squash','fútbol sala','ir a pescar');
$datos=array("edad"=>47,'localidad'=>"Pontevedra",'aficiones'=>$aficiones);
echo $nombre . ' vive en ' . $datos['localidad'] . ' y le gusta el ' . $aficiones[rand(0,count($aficiones)-1)];
echo '<br>';
echo "$nombre vive en {$datos['localidad']} y le gusta el {$aficiones[rand(0,count($aficiones)-1)]}";
echo '<br>';
echo "$nombre vive en $datos[localidad] y le gusta el {$datos['aficiones'][rand(0,count($aficiones)-1)]}";
echo '<br>';
echo "A $nombre le gusta el $aficiones[0]";
echo '<br>';
echo "$nombre tiene $datos[edad] años";
```

Funciones de manejo de arrays

Ver en <http://php.net/manual/es/ref.array.php>, pero algunas de las más importantes son:

- **count** ([mixed](#) \$array [, int \$mode = COUNT_NORMAL]) : int

Devuelve el nº de elementos del array de primer nivel. En multidimensionales, se le puede pasar un segundo parámetro para contar recursivamente.

- **array_key_exists** ([mixed](#) \$clave , array \$array) : bool ... o su alias `key_exists`

Devuelve true si la clave existe en el array, false en caso contrario.

- **array_keys** (array \$array [, mixed \$valorBusqueda = NULL]) : array

Devuelve un array numérico con las claves del array pasado en el parámetro. Opcionalmente se puede pasar un valor, devolviendo las claves que tengan dicho valor.

- **array_values** (array \$array) : array

Devuelve un array numérico con los valores del array pasado en el parámetro

- **array_merge** (array \$array1 [, array \$array2 . . .]) : array

Une los elementos de varios arrays. Si coinciden las claves de cadena, queda el último.

- **array_diff** (array \$array1 [, array \$array2 [, array \$. . .]]) : array

Devuelve un array con los valores de array1 que no están en el resto de arrays

- **implode** (string \$separador , array \$pieces) : string

- **explode** (string \$separador , string \$cadena [, int \$limite=PHP_INT_MAX]) : array

Convierte un array en una cadena con el separador indicado, y viceversa.

```
echo implode("<br>", $array);

print_r(explode('\\', 'C:\xampp\phpMyAdmin\doc\html\_images'));
Array ( [0] => C: [1] => xampp [2] => phpMyAdmin [3] => doc [4] => html [5] => _images )

print_r(explode(',', 'Pepe,María,Juan,Sonia,Alberto,Cristina'));
Array ( [0] => Pepe [1] => María [2] => Juan [3] => Sonia [4] => Alberto [5] => Cristina )
```

- **extract** (array &\$array [, int \$flags = EXTR_OVERWRITE [, string \$prefijo = **NULL**]]) : int

- **compact** ([mixed](#) \$varname1 [, [mixed](#) \$. . .]) : array

Convierte los pares clave/valor de un array asociativo en variables PHP, y viceversa. Su uso con los arrays superglobales debe hacerse con cuidado, pues puede propiciar una vulnerabilidad en la seguridad del sistema. Un buen consejo es utilizar la forma `extract(array, EXTR_PREFIX_ALL, 'prefijo')` para que un usuario malintencionado no consiga enviar el valor de una variable con un nombre usado habitualmente (\$usuario, \$contraseña, \$conexion, \$sql, etc.)

- **list**(\$var1 [, \$var2...]) = \$array

Es un constructor del lenguaje, no una función, y nos sirve para asignar múltiples variables automáticamente a los valores (ordenados y de clave numérica) de un array. Hay que tener cuidado en que haya valores suficientes para las variables especificadas, sino genera un error.

```
list($a,$b)=[1=>'1','a'=>'a',0=>'0']; echo "$a $b"; // visualiza 0 1
```

- **range** (\$inicio, \$fin [, \$incremento=1]) : array

Devuelve un array que contiene los valores indicados

- **array_map** ([callable](#) \$callback , array \$array) : array

Devuelve un array que contiene todos los elementos del array después de haberles aplicado la función pasada en el primer parámetro.

Arrays superglobales

Están definidos de tal manera que son accesibles desde cualquier lugar de cualquier script. No es ni necesario realizar un *global* para acceder a ellos dentro de funciones. Son arrays asociativos y sus claves serán, dependiendo del caso, cadenas predefinidas o nombres de otros elementos. Son los siguientes:

`$GLOBALS[]`, `$_SERVER[]`, `$_GET[]`, `$_POST[]`, `$_REQUEST[]`, `$_FILES[]`, `$_COOKIE[]`, `$_SESSION[]`, `$_ENV[]`

Aunque cada array será estudiado en la sección adecuada, comentamos ahora los dos primeros:

- `$GLOBALS[]` .- Hace referencia a las variables disponibles en el ámbito global. Las entradas de este array son los nombres de las variables globales.
- `$_SERVER[]` .- Información del entorno del servidor y de ejecución. Las [entradas de este array](#) las crea el servidor web, por lo que podrían no existir todas las esperadas.

```
echo implode("<br>", array_keys($_SERVER));

echo implode("<br>", array_values($_SERVER));
```

```
MIBDIRS
MYSQL_HOME
OPENSSL_CONF
PHP_PEAR_SYSCONF_DIR
PHPRC
TMP
HTTP_HOST
HTTP_USER_AGENT
HTTP_ACCEPT
HTTP_ACCEPT_LANGUAGE
HTTP_ACCEPT_ENCODING
HTTP_CONNECTION
HTTP_UPGRADE_INSECURE_REQUESTS
HTTP_CACHE_CONTROL
PATH
SystemRoot
COMSPEC
PATHEXT
WINDIR
SERVER_SIGNATURE
SERVER_SOFTWARE
SERVER_NAME
SERVER_ADDR
SERVER_PORT
REMOTE_ADDR
DOCUMENT_ROOT
REQUEST_SCHEME
CONTEXT_PREFIX
CONTEXT_DOCUMENT_ROOT
SERVER_ADMIN
SCRIPT_FILENAME
REMOTE_PORT
GATEWAY_INTERFACE
SERVER_PROTOCOL
REQUEST_METHOD
QUERY_STRING
REQUEST_URI
SCRIPT_NAME
PHP_SELF
REQUEST_TIME_FLOAT
REQUEST_TIME
```

```
C:/xampp/php/extras/mibs
C:/xampp/mysql/bin
C:/xampp/apache/bin/openssl.cnf
C:/xampp/php
C:/xampp/php
C:/xampp/tmp
localhost:8080
Mozilla/5.0 (Windows NT 10.0; rv:64.0) Gecko/20100101 Firefox/64.0
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
es-ES;q=0.8,en-US;q=0.5,en;q=0.3
gzip, deflate
keep-alive
1
max-age=0
C:\Program Files\Common Files\Oracle\Java\javapath;C:\ ...
C:\WINDOWS
C:\WINDOWS\system32\cmd.exe
.COM; .EXE; .BAT; .CMD; .VBS; .VBE; .JS; .JSE; .WSF; .WSH; .MSC
C:\WINDOWS
Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.2 Server at localhost Port 8080

Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.2
localhost
:::1
8080
:::1
C:/xampp/htdocs
http

C:/xampp/htdocs
postmaster@localhost
C:/xampp/htdocs/borra/index.php
59617
CGI/1.1
HTTP/1.1
GET

/borra/
/borra/index.php
/borra/index.php
1547146206.442
1547146206
```

Recuperación de los parámetros de un script

Para recuperar datos enviados desde otro script usaremos, dependiendo del método de envío de los datos, **\$_GET** o **\$_POST**. Son arrays asociativos, y sus claves se corresponden con los atributos *name* de los elementos del formulario. El array **\$_REQUEST** se completa con los valores de otros arrays superglobales (generalmente **\$_GET**, **\$_POST** y **\$_COOKIE**), vengan de donde vengan, por lo que parece más cómodo utilizarlo para no tener ni que molestarse en pensar de dónde vienen los datos, pero podría llegar a producir una vulnerabilidad del sistema por lo que no aconsejo su uso – se supone que el programador sabe por qué vía le llegan los datos, por lo que debe acceder directamente al array adecuado.

Realmente, la actualización automática del array **\$_REQUEST** depende de la directiva **request_order** del `php.ini`, a la que se le asigna una combinación de las iniciales EGPCS (de los arrays superglobales ENV, GET, POST, COOKIE y SERVER) para establecer qué datos de los arrays se copian y en qué orden. Un posible valor podría ser `request_order="GP"` con lo que se copiarían los datos de GET y los de POST.

\$_GET es especialmente útil porque nos permite obtener los datos que se envían en la propia URL, bien desde un formulario con el atributo `METHOD='GET'` (o sin atributo `METHOD`), bien desde un enlace en la propia página generado con sus parámetros adecuados. En todo caso debe utilizarse únicamente en ciertas situaciones, como ya se ha explicado en la sección correspondiente.

Como norma general, el valor será el introducido por el usuario en la propiedad *value* del elemento.

En las listas, el valor será el **value** indicado en el HTML, o el contenido del elemento en el caso de no haber *value*. En listas de selección múltiple, se podrán recuperar todos los datos asignados porque el valor es un array con todos los valores seleccionados.

En el caso de las casillas de verificación, se crearán entradas en los arrays superglobales sólo si es seleccionada – podremos comprobar su existencia con la función `isset()`. El valor introducido será, de nuevo, el contenido en el atributo *value* de la casilla.

Seguridad – filtrado de datos

Existen multitud de situaciones que pueden comprometer la seguridad de nuestro sitio web, como por ejemplo la existencia de contenidos maliciosos en arrays superglobales.

Dependiendo de para qué se va a utilizar dicho contenido utilizaremos unas u otras de las muchas funciones proporcionadas por el lenguaje.

Por ejemplo, si su contenido se va a utilizar para visualizar en pantalla, podremos utilizar las funciones **htmlspecialchars** para convertir los caracteres 'peligrosos' en sus entidades correspondientes. De esta manera, por ejemplo, podremos evitar que si el usuario introduce <...> , la función devolverá < y > evitando de esta manera que el usuario genere elementos HTML en la página, que pueden alterar su presentación (introducir ... como un nombre) o su funcionamiento (<script> </script>)

Si lo que queremos es escapar (con una barra invertida \) caracteres problemáticos a la hora de construir una cadena, podemos hacerlo con **addslashes** (escapa las comillas simples, dobles y la propia barra invertida).

También existe la función **filter_var**, muy potente y parametrizable, que nos permite filtrar una variable con un filtro entre los existentes dentro de los siguientes tipos:

- Filtros de validación: **FILTER_VALIDATE_*** EMAIL, IP, MAC, REGEXP, URL ,FLOAT ,INT, BOOLEAN
- Filtros de saneamiento: **FILTER_SANITIZE_*** EMAIL, ENCODED, MAGICQUOTES, URL, STRING, etc.

Con casi todos los filtros existen multitud de banderas que se pasan en el tercer parámetro de opciones para refinar el filtro.

Pero si el contenido se va a utilizar para construir SQL para interrogar o actualizar una base de datos, entonces no se debe utilizar addslashes, es mejor utilizar funciones propias del gestor de bd que 'escapen' los caracteres peligrosos, evitando la inyección SQL. Por ejemplo, es lo que hace la función **real_escape_string**. Aunque la mejor manera de evitar inyecciones SQL no deseadas es **parametrizar las consultas** (por nombre (con :nombre) o por posición (con ?)).

Independientemente de todas las funciones existentes en el lenguaje para sanear los datos, la utilización de expresiones regulares (función **preg_match**) nos proporcionan un método muy adecuado para filtrar hasta el límite los datos que nos interesan, asegurando su integridad y el buen funcionamiento de nuestra aplicación.

Manejo de archivos

Archivos de texto

```
<?php
if (file_exists('ruta.txt')) ...           // verificar la existencia de un archivo
    $archivo = fopen('ruta.txt', 'w') or die('Problemas al crear el archivo');
    fwrite($archivo, $texto);               // escribir texto en un archivo
    fclose($archivo);                       // cerrar el archivo

$archivo = fopen("ruta.txt", 'r') or die('Problemas al abrir el archivo');
$texto = fgets($archivo);                  // leer una línea

while(!feof($archivo)) {                  // leer línea a línea todo un archivo
    echo fgets($archivo) . '<br />';
}

$todoElTexto = file_get_contents('rutaArchivo.txt'); // lee todo el contenido
echo file_get_contents ('http://google.es');        // incluso de una URL !
file_put_contents ('archivo.txt',$texto);            // almacenar sobrescribiendo
?>
```

Subir archivos al servidor

Necesitamos, en el cliente un formulario con un tipo de **enctype=multipart/form-data**, y en la parte servidora un poco de código PHP manejando el array global **\$_FILES** (como siempre, todo puede estar almacenado en el mismo archivo...)

Asegurarse que tenemos correctamente:

- atributos enctype='multipart/form-data' y method='POST' en el formulario.
- accedemos a \$_FILES['archivo']['clave'] con las claves name, type, size, tmp_name y error.
 - name: nombre original del archivo subido en el ordenador del cliente
 - type: tipo MIME del archivo subido (si lo proporciona el navegador ...). No se asegura su certeza pues no se comprueba en el servidor.
 - size: tamaño en bytes del archivo subido
 - tmp_name: nombre establecido en la carpeta temporal del servidor
 - error: [código de error](#) producido en la subida del archivo (0==sin errores)
- usamos **move_uploaded_file** (string \$origen , string \$destino) : bool para mover el archivo a su localización definitiva.

Y los siguientes ajustes en el archivo php.ini:

- | | |
|---------------------------------|---|
| ● file_uploads=On | Para permitir subir archivos a través de HTTP |
| ● upload_tmp_dir="C:\xampp\tmp" | Carpeta temporal para los archivos subidos |
| ● upload_max_filesize = 2M | Max tamaño asignado para los archivos subidos |
| ● max_file_uploads=20 | Nº max. de archivos que subir en una petición |

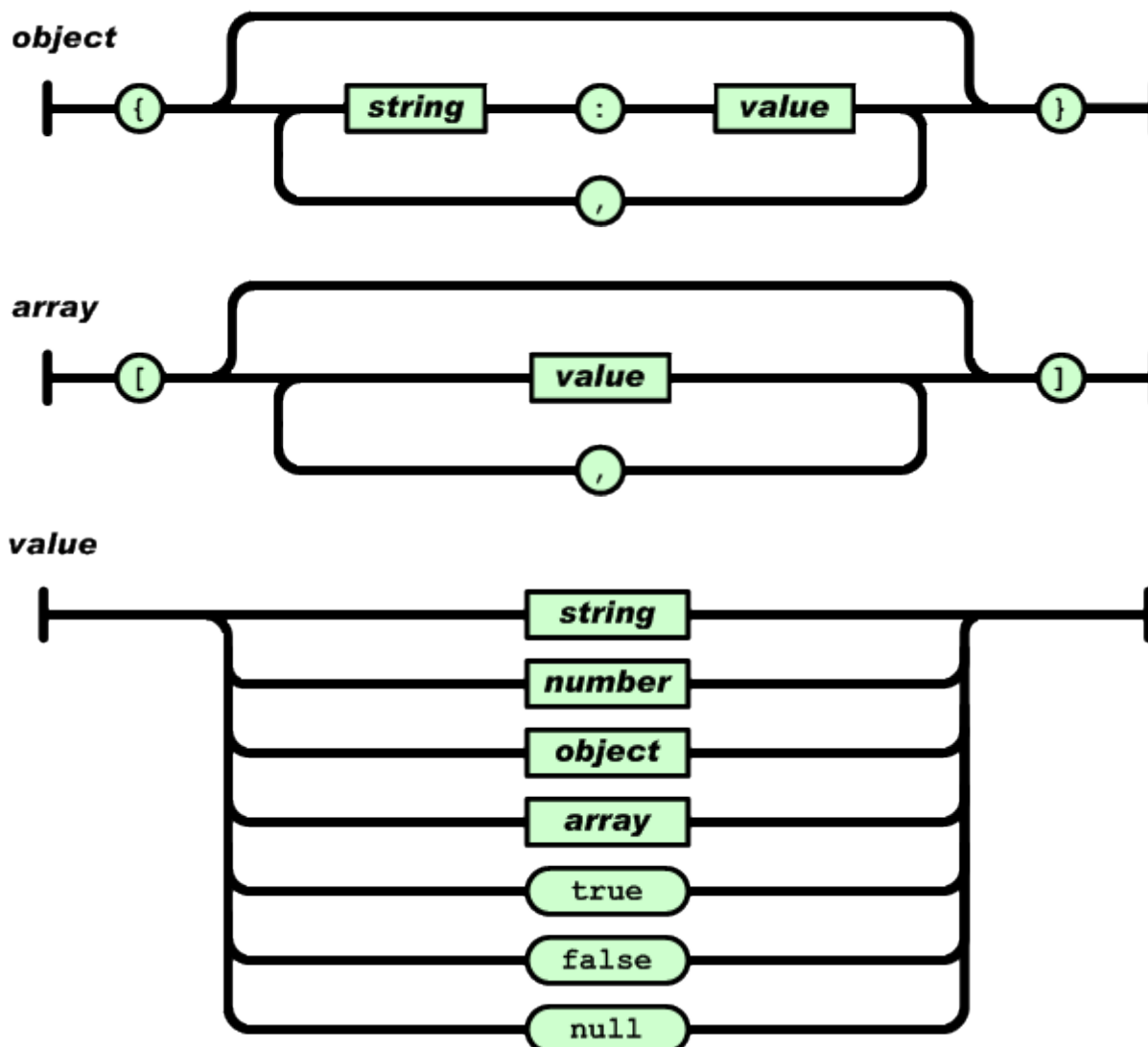
Archivos JSON

JSON (JavaScript Object Notation) es un formato de texto muy sencillo utilizado para intercambio de datos. Es una notación subconjunto de la utilizada en JavaScript, aunque hoy en día se utiliza desde cualquier lenguaje y plataforma, sobre todo como alternativa a XML.

Sus características básicas son:

- Los datos se almacenan en pares clave/valor, separados por el carácter dos puntos :
- Las claves y los valores de tipo cadena van encerradas entre comillas dobles
- Los datos se separan por comas ,
- Los arrays se almacenan entre corchetes []
- Los objetos se almacenan entre llaves { }

Fuente: <http://www.json.org/json-es.html>



Generalmente un archivo JSON ocupa menos espacio que su análogo XML, y es mucho más sencillo de analizar (en XML necesitamos un “parser” (analizador) XML en cambio para JSON basta con una sencilla función).

En PHP podremos convertir una cadena JSON en un array asociativo o en un objeto (y viceversa) de esta manera:

json_encode (\$array) : string

Devuelve la cadena que representa el array pasado como parámetro

json_decode (\$cadenaJSON[, bool \$assoc = false]) : mixed

Devuelve el resultado del análisis de la cadena JSON pasada como parámetro. Dicho resultado será un array asociativo si \$assoc==true o un objeto si \$assoc==false,

Ambas funciones tienen diferentes parámetros y opciones para indicar la profundidad a tener en cuenta a la hora del análisis, constantes como JSON_PRETTY_PRINT para formatear ‘bonito’ con espacios la cadena devuelta, escapar determinados caracteres, etc.

Configuración Apache y PHP

Directivas del servidor Apache: httpd.conf

Listen: Permite al servidor escuchar peticiones en una IP y/o puerto
Especificar una IP serviría para que no escuche en todas las IPs del servidor.

```
#Listen 12.34.56.78:80
Listen 80
```

DocumentRoot: Directorio que contiene los archivos que se servirán.
Por defecto, las peticiones se refieren a rutas en esta carpeta,
aunque existen mecanismos para alojar una web en una carpeta distinta
Nombres habituales de esta carpeta son www, wwwroot, public_html, htdocs, etc.

```
DocumentRoot "C:/xampp/htdocs"
```

DirectoryIndex: Documento por defecto
Establece el documento que devolverá el servidor cuando en la petición http viene una carpeta.
Si hay varios, se servirá el primero que se encuentre en dicha carpeta, según el orden especificado.

```
DirectoryIndex index.php index.pl index.cgi index.asp index.shtml index.html index.htm \
default.php default.pl default.cgi default.asp default.shtml default.html default.htm \
home.php home.pl home.cgi home.asp home.shtml home.html home.htm
```

Include: Nos permite incluir en el archivo de configuración el contenido de otros archivos externos
En el caso del httpd.conf configurado por XAMPP podemos observar que incluye, entre otros, el siguiente archivo

```
Include "conf/extra/httpd-xampp.conf"
```

Directory: permite especificar un conjunto de directivas de aplicación a una carpeta determinada.
Las directivas incluidas en esta sección también pueden incluirse en un archivo *.htaccess* dentro de la carpeta afectada, pero su uso debería restringirse para cuando no tengamos acceso al archivo de configuración de httpd. Esto es debido a que (si *AllowOverride* está configurado para permitir el uso de ficheros *.htaccess*), httpd buscará (y cargará si los encuentra) ficheros *.htaccess* cada vez que se solicita un documento, multiplicando las búsquedas en el sistema de archivos, penalizando el rendimiento del servidor.

Alias: permite mapear URLs a localizaciones que pueden estar fuera del sistema de archivos de DocumentRoot.

En el archivo que incluye XAMPP con sus configuraciones, encontramos que se crea una ruta especial para varias aplicaciones, entre ellas phpMyAdmin

```
Alias /phpmyadmin "C:/xampp/phpMyAdmin/"
<Directory "C:/xampp/phpMyAdmin">
...
</Directory>
```

Directivas de PHP: php.ini

Gestionar adecuadamente la configuración del PHP.INI puede evitarnos muchos problemas. Sus directivas, en principio, afectan a todos los scripts PHP del servidor por lo que, cuanto menos dependa nuestra aplicación de ellas, más fácil será la migración entre servidor de desarrollo y de producción, o menos problemas tendremos en las migraciones y/o actualizaciones futuras. Hay que pensar que no siempre tendremos permisos de administrador para modificar su configuración.

En este apartado veremos algunas de las directivas más interesantes del php.ini. Otras directivas se comentarán en otros epígrafes como las relacionadas con las sesiones o con aspectos de seguridad del sistema.

short_open_tag=Off | On

Indica a PHP si debería permitirse la forma abreviada de las etiquetas de apertura de PHP `<? ... ?>` en vez de las clásicas `<?php ... ?>`. La forma `<?=expresiónAVisualizar?>` está disponible a partir de PHP 5.4.0 independientemente de esta directiva.

output_buffering=Off|On|NumBytes

Off deshabilita el buffer de salida

On la habilita de modo ilimitado (usar con cuidado)

Especifica el nº de bytes reservado para el buffer

El uso de las funciones que gestionan el buffer de salida `ob_*`(), funcionan independientemente de la configuración existente en esta directiva: p.e. el código de abajo funciona incluso con `o_b=Off`

```
<?php
ob_start();
echo "Muuuuucho texto.";
setcookie("nombre_cookie", "datos_cookie");
ob_end_flush();
?>
```

disable_functions=

disable_classes=

Estas directivas nos permiten indicar separadas por comas, una lista de clases y/o funciones que no se podrán utilizar por razones de seguridad, por ejemplo, llamadas a comandos del sistema:

`disable_functions = "exec,system,passthru"`

max_execution_time=30

Número máximo de segundos que puede estar en ejecución cada script

max_input_time=-1|numSegundos (60)

Número máximo de segundos que puede estar el script analizando los datos de la petición http (por POST o por GET)

memory_limit=-1|cantidadDeMemoria(128M)

Máxima cantidad de memoria que puede usar un script

error_reporting sirve para indicar a PHP qué errores, avisos o notificaciones tener en cuenta para realizar una acción con ellos: enviar a un LOG, sacarlos por STDOUT, ...

Existen tres grandes tipos de errores: **ERROR:** se para el script (p.e. llamar a una función que no existe), **WARNING:** advertencias – sigue la ejecución pero probablemente haya problemas (Undefined index), y **NOTICE:** avisos – podría llegar a haber un problema (variable sin inicializar). Se puede establecer el nivel de error con la función `error_reporting($nivelDeError)` o bien utilizando esta directiva con las constantes o sus combinaciones:

`E_ALL` `E_ERROR` `E_RECOVERABLE_ERROR` `E_WARNING` `E_PARSE` `E_NOTICE` `E_STRICT` `E_CORE_ERROR`
`E_CORE_WARNING` `E_COMPILE_ERROR` `E_COMPILE_WARNING` `E_USER_ERROR` `E_USER_WARNING` `E_USER_NOTICE`
`E_DEPRECATED` `E_USER_DEPRECATED`

Valores habituales pueden ser los siguientes:

`E_ALL` (Show all errors, warnings and notices including coding standards.)

`E_ALL & ~E_NOTICE` (Show all errors, except for notices)

`E_ALL & ~E_NOTICE & ~E_STRICT` (Show all errors, except for notices and coding standards warnings.)

`E_COMPILE_ERROR|E_RECOVERABLE_ERROR|E_ERROR|E_CORE_ERROR` (Show only errors)

Valor por defecto: `error_reporting=E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED`

Valor de desarrollo: `error_reporting=E_ALL`

Valor de producción: `error_reporting=E_ALL & ~E_DEPRECATED & ~E_STRICT`

`display_errors=0n (a STDOUT) | 0ff`

Esta directiva gestiona donde PHP envía sus errores

Valores: por defecto: Off, de desarrollo On, en producción On

`log_errors=0n | 0ff`

`log_errors_max_len= 0 | numMaxLongitudErrores(1024)`

Esta directiva indica la longitud máxima de los errores almacenada en los logs

`ignore_repeated_errors= 0n | 0ff`

Esta directiva nos permite no almacenar varias veces el mismo error en el mismo script

`ignore_repeated_source= 0n | 0ff`

Esta directiva nos permite no almacenar varias veces el mismo error de scripts distintos

`post_max_size=8M`

Tamaño máximo de los datos que se aceptan por POST. 0 deshabilita el límite.

Es ignorada si la lectura de datos de POST se deshabilita a través de `enable_post_data_reading`

`default_mimetype="text/html"`

Valor que enviará PHP por defecto en el campo Content-Type de la cabecera de las respuestas http.

`default_charset="UTF-8"`

Conjunto de caracteres utilizado por defecto

`file_uploads=0n | 0ff`

Permite (o no) subir archivos

`upload_tmp_dir="C:\xampp\tmp"`

Carpeta para almacenar temporalmente los archivos

`upload_max_filesize=2M`

Tamaño máximo para cada uno de los archivos que se suben

`max_file_uploads=20`

Máximo n.º de archivos que se pueden subir en una petición

Orientación a objetos

PHP es un lenguaje de script, y originalmente carecía de muchísimas de las características de otros lenguajes modernos, como por ejemplo la orientación a objetos. Aunque a partir de la versión 4 se introdujeron algunos conceptos, es a partir de la versión 5 donde se implementaron realmente muchas de sus características importantes (no todas).

Un lenguaje es orientado a objetos cuando permite organizar los datos y su procesamiento (métodos, funciones) en clases, que abstraen su comportamiento y permiten realizar aplicaciones utilizando una metodología con multitud de ventajas (abstracción, modularidad, extensibilidad...) gracias a los mecanismos proporcionados por la POO (herencia, polimorfismo, sobrecargas, etc.)

Hoy en día programar utilizando la POO no es una ventaja, es una necesidad y lo mínimo en una aplicación profesional.

A partir de PHP 5, el modelo de objetos ha sido reescrito para tener en cuenta un mejor rendimiento y mayor funcionalidad. Este fue un cambio importante a partir de PHP 4. PHP 5 tiene un modelo de objetos completo.

Definición de una clase

La definición básica de una clase comienza con la palabra reservada **class**, seguido de un nombre de clase, y continuando con un par de llaves que encierran las definiciones de las propiedades y métodos pertenecientes a dicha clase. Los nombres de clase **no son case sensitive**.

La pseudovariable **\$this** está disponible cuando un método es invocado dentro del contexto de un objeto.

Creación de objetos

Para crear una instancia de una clase, se debe emplear la palabra reservada **new** y a continuación el nombre de la clase. Se usarán paréntesis obligatoriamente si hay que pasarle parámetros (al constructor) para inicializar el objeto. Un objeto se creará siempre, a menos que el objeto tenga un constructor que lance una excepción.

Las propiedades y métodos de una clase viven en «espacios de nombres» diferentes, por tanto, es posible tener una propiedad y un método con el mismo nombre.

Accederemos a sus miembros con **\$objeto→campo** y con **\$objeto→método()**

Asignación y Comparación de objetos

El operador de asignación en PHP (=) funciona distinto para objetos que para el resto de elementos (variables o arrays). Así como generalmente copia el valor del segundo operando en el primero, la asignación de objetos se realiza por referencia y no por valor. En el caso de querer crear un nuevo objeto con los mismos valores que otro, hay que utilizar la palabra clave **clone**.

Al utilizar el operador de comparación (==), se comparan de una forma sencilla las variables de cada objeto, es decir: Dos instancias de un objeto son iguales si tienen los mismos atributos y valores, y son instancias de la misma clase.

Pero el comportamiento del operador identidad (===) no funciona como podría parecer si tenemos en cuenta como funciona con variables (no con referencias a objetos), que comprueba el valor y el tipo de la variable. Con objetos, las variables de un objeto son idénticas sí y sólo sí hacen referencia a la misma instancia de la misma clase.

Herencia

Una clase puede heredar los métodos y propiedades de otra clase empleando la palabra reservada **extends** en la declaración de la clase. No es posible la extensión de múltiples clases, una clase sólo puede heredar de una clase base.

Los métodos y propiedades heredados pueden ser sobrescritos con la redeclaración de éstos utilizando el mismo nombre que en la clase madre. Sin embargo, si la clase madre definió un método como final, éste no podrá ser sobrescrito. Es posible acceder a los métodos sobrescritos o a las propiedades estáticas haciendo referencia a ellos con **parent::**

Constructor y destructor

Las clases que tengan un método constructor lo invocarán en cada nuevo objeto creado, por lo que se utilizará para inicializarlo antes de ser usado.

Se declara con function **__construct**, y no se debe utilizar la manera antigua de creación de constructores con el nombre de la clase.

Para ejecutar un constructor padre desde el constructor hijo se requiere invocar a **parent::__construct()**.

```
class BaseClass {  
    function __construct() { print "En el constructor BaseClass\n"; }  
}  
  
class SubClass extends BaseClass {  
    function __construct() {  
        parent::__construct();  
        print "En el constructor SubClass\n";  
    }  
}
```

Complementario a **__construct**, **__destruct** será llamado tan pronto como no hayan otras referencias a un objeto determinado, o en cualquier otra circunstancia de finalización (exit,die). También se podrá llamar a **parent::__destruct()** para llamar al destructor del padre.

Propiedades

También denominadas campos o atributos, son las variables pertenecientes a una clase. Éstas se definen usando una de las palabras reservadas `public`, `protected`, o `private`, seguido de una declaración normal de variable. Si no se incluye ninguna se asume `public`.

El uso de estas palabras reservadas indican su visibilidad: los miembros de la clase (métodos y propiedades) pueden ser:

- **public**: accesibles desde cualquier lado (en versiones antiguas se usaba **var**)
- **protected**: accesibles desde los métodos de su clase o clases descendientes
- **private**: accesibles únicamente desde los métodos de su clase

La declaración de las propiedades puede incluir una inicialización, pero esta inicialización debe ser un valor constante, es decir, debe poder ser evaluada durante la compilación y no depender de información generada durante la ejecución.

Propiedades y métodos estáticos

Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase. Una propiedad declarada como `static` no puede ser accedida con un objeto de clase instanciado (aunque un método estático sí lo puede hacer).

Debido a que los métodos estáticos se pueden invocar sin tener creada una instancia del objeto, la pseudovariable `$this` no está disponible dentro de los métodos declarados como estáticos, accediendo a través de la pseudovariable **self**, que se puede considerar como una referencia a la propia clase. El campo estático al que accedemos con `self`, incluirá `$`: (**self::\$campoEstático**)

Acceso a las propiedades

Dentro de los métodos de una clase, se puede acceder a las propiedades:

- no estáticas: con el operador `→` `$this→propiedad`
- estáticas: utilizando el operador de resolución de ámbito (doble dos puntos `::`) `self::$propiedad`

Constantes

Es posible definir valores constantes en una clase, diferenciándose de las variables en que no utilizan el símbolo `$` al declararlas o emplearlas.

```
const CONSTANTE = 'valor constante';
```

Accederemos a ellas a través del operador `::` `self::CONSTANTE` (desde los propios métodos de la clase) o `NombreClase::CONSTANTE` (desde fuera)

```

class Clase {
    const CONSTANTE=11;
    private $datoPrivado;
    public $datoPublico;
    private static $datoEstaticoPrivado=7;
    static $datoEstaticoPublico=9;

    public function __construct($dato) {
        $this->datoPrivado="$dato PRIVADO";
        $this->datoPublico="$dato PUBLICO";
    }
    function getDataPrivado() { return $this->datoPrivado; }
    static function getDataEstaticoPrivado() { return self::$datoEstaticoPrivado; }
}

$obj=new Clase("un dato");
// Todas las llamadas que hay a continuación son correctas
echo $obj->getDataPrivado();
echo $obj::$datoEstaticoPublico;
echo $obj::CONSTANTE;
echo $obj->datoPublico;
echo $obj->getDataPrivado();
echo Clase::getDataEstaticoPrivado();
echo cLaSe::getDataEstaticoPrivado();
echo $obj->getDataEstaticoPrivado();

```

Clases y métodos abstractos

Las clases definidas como abstractas no se pueden instanciar y cualquier clase que contiene al menos un método abstracto debe ser definida como tal. Los métodos definidos como abstractos simplemente declaran la firma del método, pero no pueden definir la implementación.

```

<?php
abstract class ClaseAbstracta {
    abstract protected function getValor();
}

class ClaseConcreta1 extends ClaseAbstracta
{
    protected function getValor() {
        return "ClaseConcreta1";
    }
}
?>

```


Clases y métodos finales

La palabra clave **final** impide que las clases hijas sobrescriban un método, antecediendo su definición con final. Si la propia clase se define como final, entonces no se podrá heredar de ella.

Clases anónimas

Útiles para evitar crear una nueva clase al necesitar únicamente un objeto para algo puntual.

```
<?php
$objeto->setListener(new class {
    public function f() {
        ...
    }
});
?>
```

Tienen muchas posibilidades, porque pueden heredar, implementar e incluso pasar datos a su propio constructor.

```
<?php
$objeto->setListener(new class(7, 'cadena') extends UnaClase implements UnInterface {
    public function __construct($unEntero, $unaCadena) {
        ...
    }

    public function f() {
        // implementando la función de UnInterface
        ...
    }
});
?>
```

Interfaces

Los interfaces de objetos permiten especificar qué métodos deben ser implementados por una clase, sin tener que definir cómo estos métodos son manipulados.

Son definidos utilizando la palabra clave **interface**, de la misma forma que con clases estándar, pero sin métodos que tengan su contenido definido. Todos los métodos declarados en una interfaz deben ser públicos, ya que ésta es la naturaleza de una interfaz.

Para implementar una interfaz, se utiliza el operador **implements**. Todos los métodos en una interfaz deben ser implementados dentro de la clase; el no cumplir con esta regla resultará en un error fatal. Las clases pueden implementar más de una interfaz si se deseara, separándolas cada una por una coma.

Las interfaces se pueden extender al igual que las clases utilizando el operador **extends**.

Traits

Desde PHP 5.4.0, PHP implementa una metodología de reutilización de código llamada Traits (rasgos). Son un mecanismo de reutilización de código en lenguajes de herencia simple, como PHP. El objetivo de un trait es el de reducir las limitaciones propias de la herencia simple permitiendo que los desarrolladores reutilicen a voluntad conjuntos de métodos sobre varias clases independientes y pertenecientes a clases jerárquicas distintas.

Un Trait es similar a una clase, pero con el único objetivo de agrupar funcionalidades muy específicas y de una manera coherente. No se puede instanciar directamente un Trait. Es por tanto un añadido a la herencia tradicional, y habilita la composición horizontal de comportamientos; es decir, permite combinar miembros de clases sin tener que usar herencia.

<pre><?php trait Hola { public function decirHola() { echo 'Hola '; } } trait Mundo { public function decirMundo() { echo 'Mundo'; } }</pre>	<pre><?php class MiHolaMundo { use Hola, Mundo; public function decirAdmiración() { echo '!'; } } \$o = new MiHolaMundo(); \$o->decirHola(); \$o->decirMundo(); \$o->decirAdmiración(); ?></pre>
--	--

Finalmente comentar que un trait puede componerse de otros traits:

<pre><?php trait HolaMundo { use Hola, Mundo; }</pre>	<pre>class MiHolaMundo { use HolaMundo; } ?></pre>
--	---

Sobrecarga

PHP no entiende la sobrecarga como el resto de los lenguajes OO debido a que entiende que la firma de un método es únicamente su nombre y no el número, tipo y orden de sus parámetros.

Una solución 'chapuza' es programar dicha sobrecarga implementando una única función que en función del tipo (`is_int`, `is_string`) o del número de los argumentos (`func_num_args()` y `func_get_arg($numArgumento)`) realice distintas funciones.

El concepto de sobrecarga en PHP se refiere a su capacidad de programar el comportamiento de los denominados **métodos mágicos**.

Métodos mágicos

Son métodos que permiten programar ciertas acciones cuando ocurren ciertos eventos como que se crea o se destruye un objeto, se intenta leer o asignar un campo no accesible, se intenta ver si está asignada una propiedad no accesible, etc.

Sus nombres siempre empiezan por `__` (doble underscore), y se pueden redefinir para introducir nuestras propias acciones, o simplemente para devolver el valor adecuado.

Son: `__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` y `__debugInfo()`

- **`__construct()` y `__destruct()`** Constructor y destructor, se invocan automáticamente cuando se crea y se destruye un objeto, respectivamente (ya explicados anteriormente)
- **`__get($name)` `__set($name,$value)` `__isset($name)` `__unset($name)`** Se invocan al leer o asignar, preguntar si está asignado o liberar (isset, empty, unset) propiedades innacesibles de la clase (no están incluidos sus campos private/protected).
- **`__toString()`** Permite decidir cómo comportarse cuando se trata la clase como una cadena
- **`__clone()`** Una vez finaliza la clonación (con clone), se llamará al método `__clone()` del nuevo objeto (si se ha sobrescrito) para permitirle realizar los cambios necesarios sobre sus propiedades.

```
class C {
    public $dato;
    public function __construct($dato) { $this->dato=$dato; }
    public function __isset($name) { echo "Comprobando si se asignó $name<br>"; }
    public function __get($name) { echo "Accediendo a $name<br>"; }
    public function __set($name, $value) { echo "Asignado $name a $value<br>"; }
    public function __toString() { return "El dato que tengo es $this->dato"; }
    public function __clone() { $this->dato.='-CLON'; }
}
```

```
$c=new C(11); // los paréntesis para el constructor por defecto son opcionales
$c->meInventoUnaPropiedad="cualquierCosa";
echo isset($c->meInventoUnaPropiedad)?'Asignado':'NO Asignado';
echo '<br>';
echo ($c->meInventoUnaPropiedad=='')?'Sin valor':'Tiene algo';
echo '<br>';
echo $c;
echo '<br>';
$clon=clone $c;
echo $clon;
```

La salida es:

```
Asignado meInventoUnaPropiedad a cualquierCosa
Comprobando si se asignó meInventoUnaPropiedad
NO Asignado
Accediendo a meInventoUnaPropiedad
Sin valor
El dato que tengo es 11
El dato que tengo es 11-CLON
```

Hay que tener en cuenta que cuando se clona un objeto se crea otro objeto distinto con las mismas propiedades y los mismos valores (que se copian a la nueva dirección de memoria). Pero si existen campos con referencias a otros objetos, éstas se copian pero manteniéndose apuntando a los mismos objetos, por lo que son compartidos por los dos objetos clonados.

```
class Producto {
    private $nombre;
    private $precio;
    public function __construct($nombre,$precio) {
        $this->nombre=$nombre;
        $this->precio=$precio;
    }
    public function setPrecio($precio) { $this->precio=$precio; }
    public function __toString() { return "$this->nombre a $this->precio €"; }
}

class Supermercado {
    public $nombre;
    public $productoEnOferta;
    public $productos;

    public function __construct(string $nombre, Producto $oferta, Producto ...$productos) {
        $this->nombre=$nombre;
        $this->productoEnOferta=$oferta;
        $this->productos = array();
        foreach($productos as $producto)
            $this->productos[]=$producto;
    }

    public function __toString() {
        $str="<br>$this->nombre. Hoy tenemos en oferta $this->productoEnOferta<br>";
        $str.='y también ' .implode(' ', $this->productos);
        return $str;
    }
}

$leche=new Producto("leche",1.2);
$pan=new Producto("pan",0.9);
$cerveza=new Producto("cerveza",1.4);
$colacao=new Producto("colacao",3.8);
$agua=new Producto('agua',0.2);

$supercor=new Supermercado('Supercor',$leche,$pan,$cerveza);
echo $supercor;

$corteingles=clone $supercor;
$corteingles->nombre='El corte inglés';

echo "<hr><b>Cambia precio oferta en ambas porque la referencia es la misma</b>";
$supercor->productoEnOferta->setPrecio(2);
echo $supercor; echo $corteingles;

echo "<hr><b>Meto un elemento y sustituyo otro en un array y no está en el otro porque son arrays distintos</b>";
$corteingles->productos[]=$colacao;
$corteingles->productos[1]=$agua;
echo $supercor; echo $corteingles;

echo "<hr><b>Cambio el precio del pan y cambia en ambos porque está la referencia</b>";
$corteingles->productos[0]->setPrecio(1.8);
echo $supercor; echo $corteingles;
```

La salida del ejemplo anterior sería:

```
Supercor. Hoy tenemos en oferta leche a 1.2 €
y también pan a 0.9 €, cerveza a 1.4 €

Cambia precio oferta en ambas porque la referencia es la misma      $supercor->productoEnOferta->setPrecio(2);
Supercor. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, cerveza a 1.4 €
El corte inglés. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, cerveza a 1.4 €

Meto un elemento y sustituyo otro en un array y no está en el otro porque son arrays distintos
Supercor. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, cerveza a 1.4 €
El corte inglés. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, agua a 0.2 €, colacao a 3.8 €

Cambio el precio del pan y cambia en ambos porque está la referencia  $corteingles->productos[0]->setPrecio(1.8);
Supercor. Hoy tenemos en oferta leche a 2 €
y también pan a 1.8 €, cerveza a 1.4 €
El corte inglés. Hoy tenemos en oferta leche a 2 €
y también pan a 1.8 €, agua a 0.2 €, colacao a 3.8 €
```

Por ello el uso más común de este método es capturarlo para hacer nuevas clonaciones de esas referencias internas. En este caso, introduciendo el siguiente método en la clase Supermercado, conseguimos que todos los productos (oferta y resto del array) sean nuevos objetos en memoria.

```
public function __clone() {
    $this->productoEnOferta=clone $this->productoEnOferta;
    foreach($this->productos as &$producto)
        $producto=clone $producto;
}
```

Si volvemos a ejecutar el código con esta nueva función, vemos que los cambios afectan sólo a los productos del supermercado accedido, pues no están compartidos.

```
Supercor. Hoy tenemos en oferta leche a 1.2 €
y también pan a 0.9 €, cerveza a 1.4 €
-----→      $supercor->productoEnOferta->setPrecio(2);
Supercor. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, cerveza a 1.4 €
El corte inglés. Hoy tenemos en oferta leche a 1.2 €
y también pan a 0.9 €, cerveza a 1.4 €
-----→      $corteingles->productos[]=$colacao; $corteingles->productos[1]=$agua;
Supercor. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, cerveza a 1.4 €
El corte inglés. Hoy tenemos en oferta leche a 1.2 €
y también pan a 0.9 €, agua a 0.2 €, colacao a 3.8 €
-----→      $corteingles->productos[0]->setPrecio(1.8);
Supercor. Hoy tenemos en oferta leche a 2 €
y también pan a 0.9 €, cerveza a 1.4 €
El corte inglés. Hoy tenemos en oferta leche a 1.2 €
y también pan a 1.8 €, agua a 0.2 €, colacao a 3.8 €
```

Patrón Singleton

Singleton (o instancia única) es un patrón de diseño que implementa un mecanismo de programación gracias al cual conseguimos que solo se permita una única instancia de una clase.

Para su implementación se niega el acceso al constructor creando uno de acceso privado, y evitando (de paso) la existencia de un constructor por defecto (sin parámetros).

El acceso a la instancia única se realiza a través de un método público estático que la devuelve, habitualmente denominado **getInstance()**.

```
class Singleton
{
    private static $instancia = null;

    /* Sobreescribimos los métodos mágicos __construct y __clone
       Al hacerlos privados evitamos su uso fuera de la clase.
       No se crea un constructor por defecto al existir ya uno.
    */
    private function __construct() {}

    /* hacemos lo mismo con el método mágico que se llama al clonar el objeto */
    private function __clone() {}

    public static function getInstance()
    {
        if (is_null(self::$instancia))
            self::$instancia = new Singleton();
        return self::$instancia;
    }
}

$s=Singleton::getInstance(); // correcto
$s=new Singleton();          // error
$x = clone $s;               //error
```

Acceso a BD MySQL

Introducción

El gestor de bases de datos MySQL es uno de los más populares para acceder desde PHP. Existente para multitud de plataformas, y es utilizado por algunas de las grandes empresas de la web (Facebook, Twitter, YouTube, Wikipedia, etc.)

Comprada por Sun Microsystems y ésta a su vez por Oracle, tiene una licencia dual en donde por una parte se ofrece para usos compatibles con la licencia GNU GPL, y por otra parte hay que comprarlo para productos privativos.

Otros productos alternativos de código abierto serían PostgreSQL o MariaDB (creado por el fundador de MySQL a raíz de la compra por Oracle).

La creación de bases de datos se puede realizar, como siempre, desde código o bien manualmente. Para este último caso existen diversas posibilidades, como utilizar la línea de comandos de mysql.exe si bien, con diferencia, la manera generalizada de crear y gestionar una base de datos es utilizar el interfaz web phpMyAdmin (véase el anexo correspondiente).

Para conectar y trabajar con BD desde PHP veremos dos aproximaciones: PDO y mysqli.

La extensión PDO

PDO (PHP Data Objects) existente a partir de la v.5.1, proporciona una capa de abstracción en el acceso a datos, definiendo una interfaz para poder acceder a bases de datos desde PHP. Cada controlador implementará la interfaz de la manera que considere adecuada. Así conseguimos la gran ventaja que es la independencia del código de nuestra aplicación: poder migrar la base de datos a otro gestor y el código funcionaría sin cambios (salvo el DSN – Data Source Name)

Por supuesto existen drivers PDO para acceder a las bases de datos más utilizadas: Oracle, Microsoft SQL Server, IBM DB2, Informix, MySQL, PostgreSQL, SQLite, etc (hasta 12)

También se puede consultar su existencia a través de `<?php phpinfo() ?>`

PDO

PDO support	enabled
PDO drivers	mysql, sqlite

pdo_mysql

PDO Driver for MySQL	enabled
Client API version	mysqlnd 5.0.11-dev - 20120503 - \$Id: f373ea5dd5538761406a8022a4b8a374418b240e \$

pdo_sqlite

PDO Driver for SQLite 3.x	enabled
SQLite Library	3.8.4.3

Se habilitan (descomentando, sin el ;) en el php.ini, como se puede ver a continuación:

```
...
;extension=php_pdo_firebird.dll
;extension=php_pdo_oci.dll
;extension=php_pdo_pgsql.dll
;extension=php_pdo_sqlite_external.dll
extension=php_pdo_mysql.dll
extension=php_pdo_odbc.dll
extension=php_pdo_sqlite.dll
```

La programación en PDO se hará utilizando la OOP, utilizando principalmente las siguientes clases:

```
class PDO { // representa la conexión con la BD
public __construct (string $dsn [,string $username [,string $passwd
                                [,array $options ]]])

public beginTransaction ( void ) : bool
public commit ( void ) : bool
public errorCode ( void ) : string
public errorInfo ( void ) : array
public exec ( string $statement ) : int
public getAttribute ( int $attribute ) : mixed
public static getAvailableDrivers ( void ) : array
public inTransaction ( void ) : bool
public lastInsertId ( [ string $name = NULL ] ) : string
public prepare (string $statement [, array $driver_options =
                                array()]):PDOStatement

public query ( string $statement ) : PDOStatement
public quote (string $string [,int $parameter_type=PDO::PARAM_STR ]):string
public rollBack ( void ) : bool
public setAttribute ( int $attribute , mixed $value ) : bool
}

class PDOStatement { // representa una instrucción preparada SQL y su resultado
public bindColumn (mixed $column ,mixed &$param [,int $type [,int $maxlen
                                [, mixed $driverdata]]) : bool

public bindParam (mixed $parameter,mixed &$variable [,int $data_type
                                =PDO::PARAM_STR [,int $length [,mixed $driver_options]]) : bool

public bindValue (mixed $parameter,mixed $value [,int $data_type=PDO::PARAM_STR]):bool
public closeCursor ( void ) : bool
public columnCount ( void ) : int
public debugDumpParams ( void ) : void
public errorCode ( void ) : string
public errorInfo ( void ) : array
public execute ([ array $input_parameters ] ) : bool
public fetch ([int $fetch_style [,int $cursor_orientation=PDO::FETCH_ORI_NEXT
                                [, int $cursor_offset=0]]) : mixed

public fetchAll ([int $fetch_style [,mixed $fetch_argument
                                [,array $ctor_args= array()]]):array

public fetchColumn ([ int $column_number = 0 ] ) : mixed
public fetchObject ([string $class_name="stdClass" [,array $ctor_args]):mixed
public getAttribute ( int $attribute ) : mixed
public getColumnMeta ( int $column ) : array
public nextRowset ( void ) : bool
public rowCount ( void ) : int
public setAttribute ( int $attribute , mixed $value ) : bool
public setFetchMode ( int $mode ) : bool
}
```


La extensión mysqli

Sólo sirve para trabajar con MySQL. A diferencia de PDO, ofrece un API OOP y un API procedural.

No confundir mysqli con la extensión antigua – mysql – obsoleta en PHP 5.5 y eliminada en PHP 7. La moderna es con 'i' de *improved* (mejorada). El cambio (sintáctico) básico en su uso procedural es que cambia el prefijo de las funciones (mysql_ por mysqli_), y que muchas funciones cambian un último parámetro opcional conexión, pasando a ser un primer parámetro obligatorio.

```
mysql_query("UPDATE ...", $con) >>>>>> mysqli_query($con, "UPDATE ...");
```

Al incluir los dos interfaces, siempre podemos elegir en utilizar el OOP o el procedural:

- a nivel rendimiento básicamente los dos tienen el mismo
- si es necesario migrar programas antiguos, será más rápido hacerlo a la versión procedural, por los cambios mínimos que hemos comentado anteriormente
- en el resto de los casos, siempre es más interesante programar con OOP debido a las innumerables ventajas que ésta nos ofrece.

Comparativa de características	ext/mysqli	PDO_MySQL	ext/mysql
Versión de PHP donde se introdujo	5.0	5.1	2.0
Incluida con PHP 5.x	Sí	Sí	Sí
Incluida con PHP 7.x	Sí	Sí	No
Estado de desarrollo	Activo	Activo	Mantenimiento en 5.x; eliminada en 7.x
Ciclo de vida	Activo	Activo	Obsoleto en 5.x; eliminado en 7.x
Recomendada para nuevos proyectos	Sí	Sí	No
Interfaz de POO	Sí	Sí	No
Interfaz procedimental	Sí	No	Sí
La API admite la no espera, consultas asíncronas con mysqlnd	Sí	No	No
Conexiones persistentes	Sí	Sí	Sí
La API admite conjunto de caracteres	Sí	Sí	Sí
La API admite sentencias preparadas del lado del servidor	Sí	Sí	No
La API admite sentencias preparadas del lado del cliente	No	Sí	No
La API admite procedimientos almacenados	Sí	Sí	No
API admite sentencias múltiples	Sí	La mayoría	No
La API admite transacciones	Sí	Sí	No
Las transacciones se pueden controlar con SQL	Sí	Sí	Sí
Admite toda la funcionalidad de MySQL 5.1+	Sí	La mayoría	No

*consultas asíncronas: la BD ejecuta una consulta sin esperar a que la consulta anterior finalice (no confundir con AJAX)

*mysqlnd (MySQL Native Driver–Controlador nativo de MySQL), usado por mysqli y PDO a partir de PHP 5.3.0 (no es un API que el programador utilice directamente). Antes se usaba la Biblioteca Cliente de MySQL creada por MySQL AB (ahora de Oracle) y tenía licencia MySQL por lo que en su momento se deshabilitó de forma predeterminada en PHP. Por ello se desarrolló (en C) mysqlnd como parte del proyecto PHP sin problema de licencias.

La clase `mysqli` representa la conexión entre PHP y el gestor MySQL

```
class mysqli {
    /* Propiedades */
    //nº de filas afectadas por la operación previa (INSERT,UPDATE,REPLACE,DELETE)
    int $affected_rows;
    int $connect_errno;
    string $connect_error;
    int $errno;
    array $error_list;
    string $error;
    int $field_count;
    string $client_info;
    int $client_version;
    string $host_info;
    string $protocol_version;
    string $server_info;
    int $server_version;
    string $info;
    // contiene el autoincremental generado automáticamente en la última consulta
    mixed $insert_id;
    string $sqlstate;
    int $thread_id;
    int $warning_count;
    /* Métodos */
    __construct ([string $host[,string $username[,string $passwd[,string $dbname[,int $port[,string $socket ]]]]])
    autocommit ( bool $mode ) : bool
    change_user (string $user , string $password , string $database ) : bool
    character_set_name ( void ) : string
    close ( void ) : bool
    commit ([ int $flags = 0 [, string $name ]] ) : bool
    connect([string $host[,string $username[,string $passwd[,string $dbname[,int $port[,string $socket ]]]]]) : void
    debug ( string $message ) : bool
    dump_debug_info ( void ) : bool
    get_charset ( void ) : object
    get_client_info ( void ) : string
    get_connection_stats ( void ) : bool
    mysqli_stmt::get_server_info ( void ) : string
    get_warnings ( void ) : mysqli_warning
    init ( void ) : mysqli
    kill ( int $processid ) : bool
    more_results ( void ) : bool
    multi_query ( string $query ) : bool
    next_result ( void ) : bool
    options ( int $option , mixed $value ) : bool
    ping ( void ) : bool
    public static poll (array &$read,array &$error,array &$reject,int $sec [, int $usec = 0 ] ) : int
    prepare ( string $query ) : mysqli_stmt
    query ( string $query [, int $resultmode = MYSQLI_STORE_RESULT ] ) : mixed
    real_connect ([string $host[,string $username[,string $passwd[,string $dbname[,int $port[,string $socket[,int $flags ]]]]]]) : bool
    escape_string ( string $escapestr ) : string
    real_escape_string ( string $escapestr ) : string
    real_query ( string $query ) : bool
    public reap_async_query ( void ) : mysqli_result
    public refresh ( int $options ) : bool
    rollback ([ int $flags = 0 [, string $name ]] ) : bool
    rpl_query_type ( string $query ) : int
    select_db ( string $dbname ) : bool
    send_query ( string $query ) : bool
    set_charset ( string $charset ) : bool
    set_local_infile_handler ( mysqli $link , callable $read_func ) : bool
    ssl_set ( string $key , string $cert , string $ca , string $capath , string $cipher ) : bool
    stat ( void ) : string
    stmt_init ( void ) : mysqli_stmt
    store_result ([ int $option ] ) : mysqli_result
    use_result ( void ) : mysqli_result
}
```

La clase `mysqli_result` representa el conjunto obtenido de realizar una consulta (SELECT)

```
class mysqli_result implements Traversable {
/* Propiedades */
int $current_field ;
int $field_count;
array $lengths;
int $num_rows;
/* Métodos */
data_seek ( int $offset ) : bool
fetch_all ( [ int $resulttype = MYSQLI_NUM ] ) : mixed
fetch_array ( [ int $resulttype = MYSQLI_BOTH ] ) : mixed
fetch_assoc ( void ) : array
fetch_field_direct ( int $fieldnr ) : object
fetch_field ( void ) : object
fetch_fields ( void ) : array
fetch_object ( [ string $class_name = "stdClass" [, array $params ] ] ) : object
fetch_row ( void ) : mixed
field_seek ( int $fieldnr ) : bool
free ( void ) : void
}
```

`mysqli_stmt` representa un SQL preparado (con parámetros)

```
class mysqli_stmt {
/* Propiedades */
int $affected_rows;
int $errno;
array $error_list;
string $error;
int $field_count;
int $insert_id;
int $num_rows;
int $param_count;
string $sqlstate;
/* Métodos */
__construct ( mysqli $link [, string $query ] )
attr_get ( int $attr ) : int
attr_set ( int $attr , int $mode ) : bool
bind_param ( string $types , mixed &$var1 [, mixed &$... ] ) : bool
bind_result ( mixed &$var1 [, mixed &$... ] ) : bool
close ( void ) : bool
data_seek ( int $offset ) : void
execute ( void ) : bool
fetch ( void ) : bool
free_result ( void ) : void
get_result ( void ) : mysqli_result
get_warnings ( mysqli_stmt $stmt ) : object
num_rows ( void ) : int
prepare ( string $query ) : mixed
reset ( void ) : bool
result_metadata ( void ) : mysqli_result
send_long_data ( int $param_nr , string $data ) : bool
store_result ( void ) : bool
}
```

Conectar con la base de datos

Introducción

A continuación mostramos la conexión mínima con el servidor de la BD. Este código, al ser necesario en casi todas las páginas del sitio web, no debe repetirse en cada una de ellas, sino que se situará en un archivo aparte denominado, p.e. `conexionBD.php`, incluyendo una instrucción `require('conexionBD')` al inicio.

En el caso de que nos interese conectar únicamente con el gestor de BD, no con una base de datos en particular, omitiremos el parámetro donde indicamos la base de datos. Esto puede ser necesario si, por ejemplo, vamos a ejecutar código SQL tipo “CREATE DATABASE ...”

PDO

```
<?php
$usuario='root';
$contraseña='abc123.';
$dsn = 'mysql:host=localhost;dbname=BASE_DE_DATOS';
try {
    $con = new PDO($dsn, $usuario, $contraseña);
}
catch (PDOException $ex) {
    die('No se ha podido conectar con la BD:<br/>'.$ex->getMessage());
}
?>
```

mysqli – OOP

```
<?php
$servidorBD = 'localhost';
$usuarioBD = 'root';
$passwordBD = 'abc123.';
$bd = 'BaseDeDatos';
$con = new mysqli($servidorBD, $usuarioBD, $passwordBD, $bd);

if ($con->connect_error)
    die('Problemas conectando con la BD<br>' . $con->connect_error);
?>
```

mysqli – procedural

```
<?php
$servidorBD = 'localhost';
$usuarioBD = 'root';
$passwordBD = 'abc123.';
$bd = 'BaseDeDatos';
$con = mysqli_connect($servidorBD, $usuarioBD, $passwordBD, $bd);

if (!$con)
    die('Problemas conectando con la BD<br>' . mysqli_connect_error());
?>
```

Juego de caracteres

En el caso de trabajar con distintos juegos de caracteres, puede ser interesante especificar el juego de caracteres con `$con→set_charset("utf8")` en mysqli o añadiendo `;charset=utf8` al dsn en PDO, sobre todo si vamos a almacenar los resultados obtenidos de la base de datos en cadenas JSON, pues las funciones de codificación y decodificación del lenguaje dan problemas con caracteres extraños.

También es importante tener en cuenta la posibilidad de la existencia de caracteres introducidos por usuarios o código malintencionado. En el caso de mysqli existe la función **real_escape_string** (o su alias **escape_string**) que escapa los caracteres peligrosos para evitar la temida inyección SQL. En todo caso, el uso de SQL preparado (parametrizado) ya evita en gran parte, este tipo de ataque, por lo que (sobre todo en PDO, que no existe *escape_string*) se recomienda el uso de *prepare* – (ver después).

Conexiones persistentes

Estas conexiones no son persistentes, es decir, no se reutilizan entre distintas conexiones de los scripts de un mismo sitio web. Para indicar que las conexiones sean persistentes entre distintos scripts – serán almacenadas en cache – haremos lo siguiente:

PDO: hay que establecer a true la clave PDO:ATTR_PERSISTENT del array asociativo que se pasa como cuarto parámetro en el constructor del objeto PDO.

mysqli: debe de colocar p: al nombre del host cuando se conecta.

El uso de conexiones persistentes, aunque puede mejorar mucho el rendimiento al evitar crear y destruir muchas conexiones, no evita que en algún momento la bd quede en un estado impredecible, obligando al programador a prever esas situaciones en el código.

En el archivo php.ini, en la entrada [MySQLi] existen varias directivas, como *mysqli.allow_persistent=On* para habilitar o deshabilitar el uso de conexiones persistentes, u otras para indicar el número máximo de conexiones por script y por proceso PHP.

Ejecutar SQL

En este punto nos referimos a cualquier comando SQL del tipo CREATE, INSERT, UPDATE, DELETE, ... que realizan una acción sobre la BD. Las tres aproximaciones comenzarán con:

```
require('conexionBD.php');
$sql = "INSERT INTO ...";
```

PDO	mysqli – OOP	mysqli – procedural
<pre>\$con->exec(\$sql); // devuelve las filas afectadas // si se generó un autoincrement \$id = \$con->lastInsertId();</pre>	<pre>\$con->query(\$sql); // si se generó un autoincrement \$id = \$con->insert_id;</pre>	<pre>mysqli_query(\$con, \$sql); // si se generó un autoincrement \$id = mysqli_insert_id(\$con);</pre>

Recuperar datos

Para recuperar datos de un SELECT hay que lograr un cursor a través del cual, con sucesivas llamadas a sus métodos **fetch*** nos irá devolviendo las filas seleccionadas una a una.

En función del método fetch* utilizado, recuperaremos cada fila de la consulta como:

Devuelve	mysqli	PDO
Array asociativo	fetch_assoc()	fetch() con PDO:FETCH_ASSOC
Array numérico	fetch_row()	fetchColumn() o fetch() con PDO:FETCH_NUM
Ambos	fetch_array([\$tipo=MYSQLI_BOTH])	fetch() con PDO::FETCH_BOTH
Objeto	fetch_object()	fetchObject() o fetch() con PDO::FETCH_OBJ
Todas las filas en un array	fetch_all(MYSQLI_*) MYSQLI_ASSOC/MYSQLI_NUM/MYSQLI_BOTH	fetchAll (PDO::FETCH_*)

```
require('conexionBD.php');
$sql = "SELECT ...";
```

PDO

```
$filas = $con->query($sql);
$filas->setFetchMode(PDO::FETCH_ASSOC); // ... FETCH_OBJ, FETCH_NUM, FETCH_BOTH
while($fila=$filas->fetch())
    // accedemos a $fila['campo'], $fila->campo, $fila[numCampo] ...
```

mysqli – OOP

```
$filas = $con->query($sql);
if ($filas->num_rows > 0)
    while($fila = $filas->fetch_assoc()) // _object(), _row(), _array()
        // accedemos a $fila['campo'], $fila->campo, $fila[numCampo] ...
else
    echo "No se han seleccionado registros";
```

mysqli – procedural

```

$filas = mysqli_query($sql);
if (mysqli_num_rows($filas) > 0)
    while($fila = mysqli_fetch_assoc($filas)) // _object(), _row(), _array()
        // accedemos a $fila['campo'], $fila->campo, $fila[numCampo] ...
else
    echo "No se han seleccionado registros";

```

Fijarse que los métodos **fetchAll** nos permiten obtener todos los resultados en una única llamada.

PDO – fetchAll()

```

$stmt = $con->query($sql);
$datos=$stmt->fetchAll(PDO::FETCH_OBJ);
foreach($datos as $objeto)
    echo "$objeto->campo1 $objeto->campo2";

```

La línea de fetchAll nos evita hacer lo siguiente ...

```

$datos=[];
$stmt->setFetchMode(PDO::FETCH_OBJ);
while($fila=$stmt->fetch())
    $datos[]=$fila;

```

Parametrizar el SQL

Parametrizar el SQL significa construir una plantilla de SQL para ejecutarla múltiples veces cambiando únicamente los parámetros (los datos variables). Dichos parámetros se incluyen en el SQL bien con un ? (parámetro por posición) o con un :nombreParámetro (parámetro por nombre).

Mysql permite sólo por posición, PDO en cambio admite ambos.

Una vez el SQL está construido se 'prepara' para que el gestor de BD la interprete, la compile, la optimize y la almacene para ser utilizada posteriormente cuando le lleguen los parámetros. Finalmente, le asignaremos distintos valores a los parámetros, ejecutando el SQL cuantas veces sea necesario.

Las ventajas de este método básicamente son tres:

- la consulta se interpreta una única vez
- se minimiza el ancho de banda al enviar únicamente los parámetros
- se minimiza la posibilidad de inyección SQL (escapando/entrecorriendo automáticamente)

La desventaja sería que una sentencia preparada para ejecutarse una sola vez causa más viajes de ida y vuelta desde el cliente al servidor que una sentencia no preparada

PDO

```

$sql="INSERT INTO productos(campo1,campo2,campo3) VALUES (:campo1,:campo2,:campo3)";
$consultaPreparada = $con->prepare($sql);

$consultaPreparada->bindParam(':campo1',$campo1); // con ? bindParam(1,$campo1)
$consultaPreparada->bindParam(':campo2',$campo2); // con ? bindParam(2,$campo2)
$consultaPreparada->bindParam(':campo3',$campo3); // con ? bindParam(3,$campo3)

$campo1 = 57; $campo2 = "Valor campo registro 1"; $campo3 = 4.5;
$consultaPreparada->execute();

$campo1 = 23; $campo2 = "Valor campo registro 2"; $campo3 = 1.1;
$consultaPreparada->execute();

$campo1 = 34; $campo2 = "Valor campo registro 3"; $campo3 = 2.3;
$consultaPreparada->execute();
$consultaPreparada=null;

```

mysqli – OOP

```

$sql="INSERT INTO tabla(campo1,campo2,campo3) VALUES (?, ?, ?)";
$consultaPreparada = $con->prepare($sql);
$consultaPreparada->bind_param("isd", $campo1, $campo2, $campo3);
// i ~ integer d ~ double s ~ string b ~ BLOB

$campo1 = 57; $campo2 = "Valor campo registro 1"; $campo3 = 4.5;
$consultaPreparada->execute(); // filas afectadas con ->affected_rows;

$campo1 = 23; $campo2 = "Valor campo registro 2"; $campo3 = 1.1;
$consultaPreparada->execute();

$campo1 = 34; $campo2 = "Valor campo registro 3"; $campo3 = 2.3;
$consultaPreparada->execute();

$consultaPreparada->close();
$con->close();

```

mysqli – procedural

```

$sql="INSERT INTO tabla(campo1,campo2,campo3) VALUES (?, ?, ?)";
$consultaPreparada = mysqli_prepare($con,$sql);
mysqli_stmt_bind_param($consultaPreparada, "isd", $campo1, $campo2, $campo3);
// i ~ integer d ~ double s ~ string b ~ BLOB

$campo1 = 57; $campo2 = "Valor campo registro 1"; $campo3 = 4.5;
mysqli_stmt_execute($consultaPreparada);

$campo1 = 23; $campo2 = "Valor campo registro 2"; $campo3 = 1.1;
mysqli_stmt_execute($consultaPreparada);

$campo1 = 34; $campo2 = "Valor campo registro 3"; $campo3 = 2.3;
mysqli_stmt_execute($consultaPreparada);

mysqli_stmt_close($consultaPreparada);

```


PDO

```

$sql="SELECT * FROM tabla WHERE campo1=:campo1 AND campo3=:campo3";
$consultaPreparada = $con->prepare($sql);

$consultaPreparada->bindParam(':campo1',$campo1,PDO::PARAM_INT);
$consultaPreparada->bindParam(':campo3',$campo3); // con ? bindParam(2,$campo3)

$campo1 = 57;    $campo3 = 4.55;
$consultaPreparada->execute();

echo $consultaPreparada->rowCount() . " fila/s afectada/s";
while($fila = $consultaPreparada->fetch(PDO::FETCH_ASSOC))
    // accedemos a $fila['campo1'] ...

$consultaPreparada->closeCursor();
$consultaPreparada=null;

```

mysqli – OOP

```

$sql="SELECT * FROM tabla WHERE campo1=? AND campo3=?";
$consultaPreparada = $con->prepare($sql);
$consultaPreparada->bind_param("id", $campo1, $campo3);
// i ~ integer    d ~ double    s ~ string    b ~ BLOB

$campo1 = 57;    $campo3 = 4.5;
$consultaPreparada->execute();

$filas = $consultaPreparada->get_result();
echo $filas->num_rows . " fila/s afectada/s";

while($fila = $filas->fetch_assoc()) // _object(), _row(), _array()
    // accedemos a $fila['campo'], $fila->campo, $fila[numCampo] ...

$consultaPreparada->close();

```

mysqli – procedural

```

$sql="SELECT * FROM tabla WHERE campo1=? AND campo3=?";
$consultaPreparada = mysqli_prepare($con,$sql);
mysqli_stmt_bind_param($consultaPreparada, "id", $campo1, $campo3);
// i ~ integer    d ~ double    s ~ string    b ~ BLOB

$campo1 = 57;    $campo3 = 4.5;
mysqli_stmt_execute($consultaPreparada);

$filas = mysqli_stmt_get_result($consultaPreparada);
echo mysqli_num_rows($filas) . " fila/s afectada/s";

while($fila = mysqli_fetch_assoc($filas)) // _object(), _row(), _array()
    // accedemos a $fila['campo'], $fila->campo, $fila[numCampo] ...

mysqli_stmt_close($consultaPreparada);

```

Transacciones

Para poder realizar transacciones necesitamos al menos PHP 5.5.0, MySQL 5.6 y, por supuesto, un motor de almacenamiento en el gestor de BD que las permita (InnoDB sí, MyISAM no).

Las transacciones ofrecen cuatro características principales: Atomicidad, Consistencia, Aislamiento y Durabilidad (ACID). De esta manera, se garantiza que cualquier trabajo llevado a cabo en una transacción, incluso si se hace por etapas, sea aplicado a la base de datos de forma segura, y sin interferencia de otras conexiones (al hacer un COMMIT). El trabajo de la transacción puede deshacerse (con un ROLLBACK), siempre que no se haya ejecutado previamente un commit, lo cual facilita deshacer los cambios realizados hasta el momento al detectar algún tipo de contratiempo.

Por defecto, MySQL trabaja con el modo *autocommit* activado, o sea, que toda consulta que se ejecute tiene su propia transacción implícita, si la base de datos la admite, o ninguna transacción si la base de datos no las admite. Eso significa que en el momento en que se ejecuta un SQL que modifica una tabla, el cambio se actualiza en disco haciéndolo permanente, sin posibilidad de vuelta atrás.

Para realizar una transacción, básicamente ejecutaremos un **beginTransaction**, ejecutaremos el código necesario y finalmente, realizaremos un **commit** para realizar los cambios o un **rollback** para deshacerlos.

Si un script finaliza o la conexión está a punto de ser cerrada, si existe una transacción pendiente se revierte automáticamente. Esto es una medida de seguridad que ayuda a evitar inconsistencias en los casos donde el script finaliza inesperadamente (si se no confirmó la transacción, se asume que algo salió mal, con lo cual se realiza la reversión para la seguridad de los datos).

PDO

```
$bdd->setAttribute(PDO::ATTR_AUTOCOMMIT,0);  
$con->beginTransaction();    $con->commit();    $con->rollBack();
```

mysqli – OOP

```
$con->autocommit(FALSE);  
$con->begin_transaction();    $con->commit();    $con->rollback();
```

mysqli – procedural

```
mysqli_autocommit($con, FALSE);  
mysqli_begin_transaction($con); mysqli_commit($con); mysqli_rollback($con);
```

Cerrar la conexión

PDO	mysqli – OOP	mysqli – procedural
<code>\$con = null;</code>	<code>\$con->close();</code>	<code>mysqli_close(\$con);</code>

Las conexiones se cerrarán de la manera indicada anteriormente aunque, de no realizarse explícitamente, PHP cerrará la conexión automáticamente al finalizar el script si no se ha indicado que las conexiones sean persistentes (ver el epígrafe anterior de conexión con el servidor).

Mantenimiento del Estado HTTP – Cookies y Sesiones

Introducción

HTTP es un protocolo de naturaleza desconectada, o sin estado, puesto que no proporciona ningún método para que el servidor pueda distinguir las peticiones de los clientes. Las distintas peticiones (HTTP request) no tienen ninguna relación entre ellas, simplemente sirve las páginas cada vez que se las piden.

Este comportamiento tiene un beneficio inmediato en el rendimiento, puesto que mantener el estado de las conexiones aumenta el ancho de banda consumido, pero delega en la programación resolver el problema de que una aplicación mantenga el estado de cada cliente conectado: p.e. saber si está autenticado, persistencia del carrito de la compra, etc.

Una primera aproximación para identificar a los usuarios, y con ello poder establecer el concepto de sesión, es pasar un identificador de sesión en la propia URL, utilizando el mecanismo GET. Aunque muy cómodo no siempre es lo más adecuado por sus conocidos problemas de seguridad, además de la cantidad y tipo de los datos que se pueden pasar en la propia URL.

Cookies

Las cookies: pequeñas cantidades de datos que envía un servidor web al cliente web, y que éste almacena, para enviarle de nuevo al servidor web en futuras conexiones. Aunque se pueden utilizar para muchos fines, podemos decir que existen dos grandes funcionalidades:

Las cookies persistentes o permanentes: muy cómodas para datos con poca relevancia: último día conectado a la web, idioma elegido, secciones visitadas, etc. Tienen interés para fines de análisis de navegación, personalización de la web, publicidad ... Tienen una fecha de caducidad o expiración determinada, y pasado ese tiempo el navegador las elimina.

Las cookies de sesión, que son temporales (en memoria) para que no se almacenen en disco evitando ser leídas por terceros para posibles ataques de suplantación de identidad.

Las cookies en principio tampoco solucionan el problema de la seguridad, puesto que, por ejemplo, son manipulables en el cliente. Además se pueden deshabilitar en el cliente, por lo que si nuestra web no tiene un mecanismo alternativo (vía GET en la URL), no funcionaría correctamente.

Las cookies las gestiona el navegador que las recibe y las almacena como crea más conveniente, por lo general como archivos en una carpeta local.

Para implementar las cookies se crearon dos cabeceras HTTP:

- 'Cookie' en la cabecera de la petición (request)
- 'Set-Cookie' en la cabecera de la respuesta (response).

Así, cuando un cliente hace una primera petición (request) a un servidor web, éste si quiere puede incluir en la respuesta (response) una cabecera 'Set-Cookie'. Así, el navegador incluirá una cabecera 'Cookie' automáticamente en las futuras peticiones (request) a ese servidor web.

Cliente	>>>	HTTP request	>>>	Servidor
Cliente	<<<	HTTP response + Set-Cookie	<<<	Servidor
Cliente	>>>	HTTP request + Cookie	>>>	Servidor
Cliente	<<<	HTTP response	<<<	Servidor
Cliente	>>>	HTTP request + Cookie	>>>	Servidor

Para crear una cookie utilizaremos la función *setcookie*:

```
bool setcookie(
    string $name           nombre de la cookie
    [,string $value        valor de la cookie
    [,int $expire = 0      tiempoEnQueCaduca ... time() +- segundos o 0==finDeSesión
    [, string $path        ruta en el servidor en que está disponible la cookie
    [, string $domain       dominio en el que la cookie estará disponible
    [, bool $secure = false solo se transmite por HTTPS
    [, bool $httponly = false ]]]])sólo accesible por HTTP, no en JavaScript
```

Devuelve FALSE si ya hubo alguna salida (output) anterior, TRUE si se ejecutó correctamente, aunque no podemos saber si fue aceptada en el cliente.

Finalmente, para comprobar la existencia y, en su caso, acceder al valor de una cookie almacenada en el cliente, utilizaremos el array asociativo `$_COOKIE[nombreDeLaCookie]`.

Sesiones

Si el servidor, cuando un cliente accede a un sitio web, genera un identificador aleatorio y único para incluir en las futuras cabeceras de las comunicaciones HTTP, se podrían relacionar dichas conexiones y mantener el estado de cada conexión de manera independiente en el servidor. Para ello utiliza las denominadas cookies de sesión, y que por lo general el navegador las mantiene en memoria, pues no tiene sentido almacenarlas debido a que se eliminan al terminar la navegación por dicha web.

La propagación del identificador de sesión se puede realizar de dos maneras:

- A través de cookies. Es el método más seguro, pero no siempre están disponibles pues pueden deshabilitarse en el navegador
- En la propia URL, más inseguro que a través de cookies.

De esta manera nace el concepto de sesión, que podría definirse como el estado que mantiene el servidor web para un cliente determinado, desde que se conecta por primera vez hasta que finaliza su navegación, diferenciándolo del resto de sus otras conexiones a través de los métodos comentados anteriormente. Es decir, la navegación que realiza un usuario sobre las distintas páginas de una misma web hasta que cierra el navegador o cambia de web sin volver en un tiempo determinado a la anterior.

Gracias a este mecanismo, al poder diferenciar el servidor cada conexión, puede almacenar información en su memoria para ser compartida y manipulada en posteriores accesos por otras páginas de la misma aplicación web y conexión.

Debido a que la implementación del mantenimiento de la sesión se hace a través de mecanismos que dependen del navegador (URL/cookie), para establecer dos conexiones distintas desde el mismo equipo a la misma web tenemos las siguientes opciones:

- Utilizar navegadores distintos.
- Utilizar distintos perfiles de Chrome
- Utilizar la ventana de incógnito o privada del navegador: se ejecuta en una ventana separada de las sesiones normales de navegación.
- En Google, por ejemplo, podemos dar de alta distintas cuentas en un único perfil y cambiar de cuenta en la misma ventana del navegador, pero esto se puede realizar porque la aplicación que se ejecuta en el servidor ha sido programada para realizar esta tarea: realmente es como un nuevo *login* dinámico, no una conexión distinta.

Obviamente, cuando se incrementa el número de usuarios (de conexiones), el uso de la memoria del servidor se multiplica, por lo que hay que almacenar la menor cantidad posible de información para no llegar a una situación en donde el servidor se quede sin memoria disponible.

Configuración PHP.INI de la sesión

El archivo *php.ini* es el archivo de configuración de PHP. En él podemos encontrar multitud de directivas que se aplican a multitud de aspectos del funcionamiento del intérprete de PHP.

Podemos cambiarlas editando directamente en dicho archivo y afectando a todos los scripts PHP del servidor, o establecer su valor durante la ejecución del script utilizando *ini_set(\$directiva, \$valor)*. Aunque no todos los servidores lo permiten y no todas las directivas son accesibles desde esta función, es interesante cuando estamos en un servidor compartido donde no tengamos permisos para realizar cambios a los archivos de configuración. El nuevo valor se mantendrá durante la ejecución del script y se restaurará al finalizar.

Opcionalmente también se pueden establecer dichas directivas a través de un array asociativo pasándoselas como parámetro a la función *session_start*, que veremos posteriormente.

session.use_cookies

por defecto 1, para gestionar el ID de sesión a través de una cookie

session.use_trans_sid

por defecto 0. Utilizar con mucha precaución, pues permite enviar de manera transparente el ID de sesión en la URL. Fácilmente accesible, podrían suplantar nuestra identidad fácilmente.

session.use_only_cookies

estableciéndolo a 1 forzamos a PHP a usar únicamente cookies para almacenar el ID de sesión

session.name=PHPSESSID

nombre de la sesión (~nombre cookie), podremos establecerlo con *session_name*

session.auto_start

activando esta directiva se inicia la sesión automáticamente en cada petición

session.cookie_lifetime

segundos de vida de la cookie, si es 0 permanece hasta que se cierra el navegador

session.cookie_httponly

activando esta directiva, el navegador no podrá acceder a ella a través de Javascript, lo cual aumenta la seguridad al evitar en cierta manera el cross site scripting (XSS)

session.cookie_secure

activándolo nos aseguramos que las cookies se envíen sólo sobre conexiones seguras

session.cache_expire

especifica los minutos de vida para las páginas de sesión examinadas. Por defecto 180.

Funciones de gestión de la sesión

PHP tiene soporte para sesiones, igual que el resto de los lenguajes de programación de servidor, y proporciona métodos para gestionar su manejo de una manera sencilla y transparente:

string session_name([string \$name])

Devuelve el nombre de la sesión actual, y opcionalmente se le puede pasar como parámetro el nuevo nombre para la sesión actual (el valor por defecto lo podemos ver en el php.ini). Hay que llamarlo antes de `session_start()`, y es utilizado para darle nombre a la cookie (deberían ser caracteres alfanuméricos), que contendrá como valor el `session_id()`.

Es interesante establecerlo en cada aplicación web si comparten máquina (o declarar correctamente los VirtualHost adecuados en el http.conf) para no compartir los datos de sesión con el mismo nombre, en caso contrario no tiene especial interés el modificarlo.

bool session_start([array \$options=[]])

Inicia una nueva sesión o reanuda la existente. No se debe llamar si el script ya ha generado alguna salida (HTML estático, echo, etc.)

Opcionalmente se le puede pasar un array asociativo con valores que sobrescribirán las directivas de configuración de sesiones establecidas en el php.ini

Una llamada a `session_start()` realiza lo siguiente:

- Comprueba si ha llegado un PHPSESSID en una cabecera Cookie de la petición, o en su defecto, si ha llegado como parámetro GET en la URL.
- En cualquiera de ambos casos, llena el array superglobal `$_SESSION` con los datos almacenados hasta el momento para la sesión del identificador obtenido.
- En caso contrario genera un nuevo PHPSESSID para su uso posterior.
- Hace un `setCookie` para enviar en la cabecera de la respuesta el PHPSESSID

A partir del momento en que llamamos a `session_start()`, ya podemos acceder al array asociativo `$_SESSION` añadiendo, eliminando o modificando sus entradas.

```
$_SESSION['usuario']=$usuario; // creamos o actualizamos una variable de sesión  
unset($_SESSION['usuario']); // eliminamos una variable de sesión
```


string session_id([string \$id])

Obtiene y/o establece el ID de sesión actual.

void session_write_close(void)

Aunque las sesiones se cierran automáticamente al terminar el script, se pueden cerrar manualmente llamando a esta función.

void session_unset(void)

Elimina todas las variables de sesión, es decir, todas las entradas del array `$_SESSION`. Debe llamarse a esta función y no hacer `unset($_SESSION)` porque ésto hará que no se pueda seguir utilizando dicho array.

bool session_destroy(void)

Destruye toda la información asociada con la sesión actual, pero no destruye las variables globales asociadas con la sesión ni destruye la cookie de sesión, por lo que llamando de nuevo a `session_start()` se podrían volver a utilizar. Devuelve `TRUE` o `FALSE` indicando si se destruyó satisfactoriamente.

Se pueden consultar en este mismo manual distintos ejemplos donde se utilizan sesiones. En concreto, cualquier aplicación web que requiera autenticación (login) necesita utilizar sesión para almacenar, como mínimo, el usuario.

Protocolo HTTP

HTTP (Hypertext Transfer Protocol) es un protocolo a nivel de aplicación que permite las transferencias de información en la WWW.

En 1991 sale la primera versión HTTP/0.9 era muy básica (sólo disponía de GET) y sólo podía transferir datos sin formato a través de internet.

En 1996 sale la versión HTTP/1.0 (RFC 1945) mejoró el protocolo permitiendo mensajes con formato (MIME), conteniendo metadatos sobre los datos transferidos y otras mejoras, aunque con grandes carencias como pocos códigos de estado, baja calidad de las autentificaciones, problemas con los proxy's, etc.

En 1999 sale la versión HTTP/1.1 que añade, entre otras, las siguientes características:

- soporte de host's virtuales (diferentes nombres de dominio alojados bajo una misma IP)
- capacidad de recuperación de transferencias interrumpidas
- soporte para transferencias fragmentadas (*chunked transfer*), un sistema de streaming que permite enviar la información en fragmentos (cada uno precedido de su tamaño en bytes). Esto permite ir enviando información sin saber su tamaño total, lo cual repercute en una respuesta más rápida en las páginas generadas dinámicamente.
- soporta conexiones persistentes, lo cual permite enviar varias peticiones (request) bajo una misma conexión reduciendo así tiempos de respuesta y ancho de banda al no tener que crear la misma conexión continuamente.

A partir de 2015 surge la última versión del protocolo, la HTTP 2.0, y que las últimas versiones de los navegadores ya comienzan a soportar, siendo además compatible con HTTP 1.1.

El protocolo HTTP se basa en peticiones (request) del cliente al servidor y respuestas (response) del servidor al cliente.

Mensajes HTTP

Los mensajes HTTP pueden ser peticiones (request) del cliente al servidor o respuestas (response) del servidor al cliente. Ambos mensajes tienen la misma estructura:

<p style="text-align: center;">PrimeraLínea [Cabeceras generales, de entidad y/o específicas CRLF] CRLF [Cuerpo del mensaje]</p>

donde la primera línea depende de si es un request o un response, y CRLF es la combinación de CR (carriage return, retorno de carro, hex 0D ~13) y LF (line feed, salto de línea, hex 0A ~10)

Las cabeceras tienen el formato `nombreCabecera:valorCabecera`, y pueden agruparse en:

- **generales:** se aplican tanto a peticiones como a respuestas, pero no tienen relación con los datos que se puedan transmitir en el cuerpo
- **de entidad:** contienen información añadida del cuerpo, como su tamaño o tipo MIME
- **de petición:** contienen información sobre el recurso pedido o sobre el propio cliente
- **de respuesta:** contienen información sobre la respuesta o sobre el propio servidor (nombres, versiones, etc).

Las cabeceras generales (aplicables tanto a peticiones como a respuestas) son las siguientes:

Cache-Control – se usa para indicar directivas que deben ser cumplidas por los mecanismos de cacheo (sobreescriben su comportamiento predeterminado). Su uso es unidireccional, por lo que su uso en una petición no implica que es aplique a su respuesta, y viceversa.

Connection – permite enviar opciones que queremos enviar en esta conexión particular, pero que no debe ser comunicada por los proxies en futuras conexiones. Por ejemplo, la cabecera **Connection: close** se envía para que, después de generar la respuesta, se cierre la conexión actual (no se considere persistente). Las aplicaciones que no soporten conexiones persistentes, lo tendrán que enviar en cada mensaje.

Date – indica la fecha y hora a la que se originó el mensaje

Pragma –

Trailer –

Transfer-Encoding – indica (si existe) qué tipo de transformación ha sido aplicada al cuerpo del mensaje para ser transmitida entre el origen y el destino.

Upgrade – lo usa el cliente para indicar qué otros protocolos de comunicación soporta y quiere usar, si el servidor puede hacerlo. El servidor debe usar este campo si genera una respuesta 101 **Switching Protocols** para indicar a qué protocolo cambia, p.e.

Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
--

Via – usado por proxies y gateways para indicar los protocolos intermedios que se deben usar en las peticiones y en las respuestas.

Warning – se usa para transportar, en lenguaje natural, información adicional acerca del estado del mensaje.

Las **cabeceras de entidad** (aplicables tanto a peticiones como a respuestas) son las siguientes:

Allow – especifica los métodos soportados por la URI (recurso) de la petición.

Allow: GET, HEAD, PUT

Content-Encoding – informa de la codificación utilizada en el contenido de la cabecera **Content-Type**. Se usa generalmente para permitir comprimir un recurso sin perder información, p.e.

Content-Encoding: gzip

Content-Language – describe el idioma del contenido, para poder filtrar contenidos en función del perfil del usuario

Content-Length – indica el tamaño del contenido en bytes

Content-Location – describe un URI distinto del URI de la petición, donde se puede encontrar el recurso en el momento actual.

Content-MD5 – es el MD5 del contenido, que se puede utilizar para chequear su integridad.

Content-Range –

Content-Type – indica el tipo del cuerpo enviado, p.e.

Content-Type: text/html; charset=ISO-8859-4

Expires – indica la fecha y hora a partir de la cual la respuesta se considera antigua

Last-Modified – Indica la última fecha y hora de modificación del recurso. Dependiendo de la naturaleza del recurso, el servidor intentará deducir su valor: si es un archivo, será la última modificación del sistema de archivos, si es un contenido dinámico, será la fecha más moderna de entre las de sus componentes, si es de una BD, será el último timestamp del registro, etc.

Peticiones HTTP (Request)

Una petición al servidor tiene la siguiente forma:

```

Método URI-recurso VersiónHTTP
[ Cabeceras (generales, de entidad o de petición) CRLF ]
CRLF
[ Cuerpo del mensaje ]

```

URI-recurso

Identifica el recurso sobre el que actuará el método. Se usará un * cuando la petición no se realice sobre un recurso en concreto sino sobre el servidor en sí.

Método (verbo)

Especifica la acción que se realizará en el recurso. Se consideran métodos **idempotentes** aquellos cuyos efectos son los mismos realizando 1 o más veces una misma petición.

GET – Únicamente recupera información, sin ningún efecto colateral sobre los datos

HEAD – Es exactamente igual a GET, pero únicamente se devuelve la cabecera, no el contenido

POST – Envía datos al servidor para su procesamiento (inserción o actualización). Los datos viajan en el cuerpo de la petición.

PUT – Reemplaza la representación del recurso por el nuevo contenido. La diferencia con POST es su idempotencia.

PATCH – Similar a PUT pero para realizar modificaciones parciales. No es idempotente.

DELETE – Elimina la representación del recurso indicado en la URI

OPTIONS – Describe las opciones disponibles para la comunicación (p.e. los métodos permitidos por el servidor)

TRACE – Realiza un mensaje de test sobre la ruta existente hasta el recurso. En concreto, se solicita al servidor que incluya en la respuesta todos los datos que recibe en la petición, que puede tener información que han incluido los servidores intermedios.

CONNECT – Sirve para establecer una comunicación bidireccional con el servidor indicado en la URI. Se usa por ejemplo con un proxy para realizar un tunel con SSL.

Los métodos idempotentes son GET, HEAD, PUT, DELETE, OPTIONS y TRACE.

Cabeceras de petición

Accept-Charset – indica qué charset's se aceptan para la respuesta, p.e.

Accept-Charset: utf-8, iso-8859-1;q=0.5 (q es el peso (la prioridad) del elemento, 1 si no se indica)

Accept-Encoding – indica las codificaciones aceptables del contenido. Puede ser gzip (LZ77 – Lempel-Ziv coding), compress (LZW), deflate (algoritmo de compresión), br (algoritmo Brotli), identity (sin compresión, siempre aceptable), etc.

Accept-Encoding: compress, gzip (ambos tienen la misma prioridad, 1.0)

Accept-Encoding: compress;q=0.5, gzip;q=1.0 (preferiblemente gzip, 0 indicaría inaceptable)

Accept-Language – indica los lenguajes naturales que se aceptan para la respuesta, p.e.

Accept-Language: da, en-gb;q=0.8, en;q=0.7 indica que acepta preferiblemente el danés, luego el inglés británico y finalmente cualquier otro tipo de inglés

Authorization – Generalmente (no necesariamente) después de recibir un 401 Unauthorized y posiblemente una respuesta WWW-Authenticate, con esta cabecera indicamos las credenciales que contienen la información de autenticación.

Authorization: Basic YWxhZGRpbjpvGVuc2VzYW1l base64(usuario:password)
no es un método de cifrado, solo de compresión. Debe ir sobre HTTPS

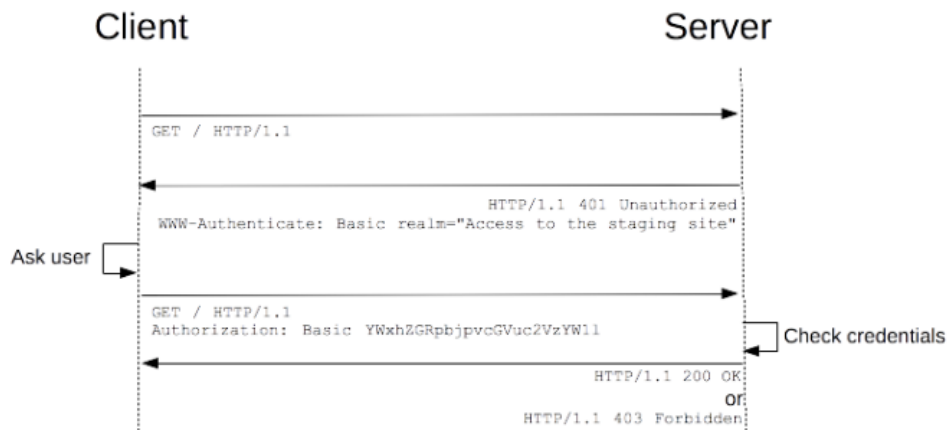


Imagen obtenida en <https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication>

Cookie – el cliente envía una cookie al servidor que previamente éste le había enviado en una cabecera Set-cookie. El envío se decide en función del servidor de la petición, la URI y la antigüedad de la cookie.

Expect –

From – contiene una dirección de correo electrónico a la que poder recurrir si se detecta algún problema, o bien almacenar en algún log.

Host – especifica el host y el puerto donde se pedirá el recurso, p.e.

```
GET /pub/WWW/ HTTP/1.1
```

```
Host: www.w3.org
```

If-Match – **If-Modified-Since** – **If-None-Match** – **If-Range** – **If-Unmodified-Since** –

Max-Forwards – se indica un número máximo de reenvíos para proporcionar un mecanismo de limitación a los métodos TRACE y OPTIONS. Cada proxy o gateway decrementa el número y no lo reenvía al llegar a cero.

Proxy-Authorization – contendrá las credenciales para autenticarse contra el proxy

Range –

Referer – permite al cliente especificar, por si le viene bien al servidor, el URI desde el cual se ha obtenido la URI de la petición (no se enviará si la URI pedida se obtuvo de un origen que no tiene su propia URI, p.e. un usuario que la ha tecleado)

TE –

User-Agent – contiene información sobre la aplicación (y subproductos) que originaron la petición

```
Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0
```

```
Mozilla/5.0 (Macintosh; Intel Mac OS X x.y; rv:42.0) Gecko/20100101 Firefox/42.0
```

Respuestas HTTP (Response)

Una respuesta del servidor tiene la siguiente forma:

<p style="text-align: center;"> VersiónHTTP CódigoDeEstado Descripción [Cabeceras (generales, de entidad o de respuesta) CRLF] CRLF [Cuerpo del mensaje] </p>
--

Código de estado y Descripción

Entero de tres dígitos con el resultado de la respuesta generada a la petición. La descripción es una cadena de texto legible para un humano, y que resume dicho código de estado.

Los códigos de estado HTTP se dividen en cinco categorías:

- 1xx Respuestas informativas. Se recibió la petición.
- 2xx Éxito. La acción fue recibida, entendida y aceptada
- 3xx Redirecciones. Es preciso alguna acción para completar la petición.
- 4xx Errores del cliente. La petición tiene algún error sintáctico, o no se puede realizar.
- 5xx Errores del servidor. La petición es válida pero el servidor ha fallado al atenderla.

A continuación se muestran los estados de error con su descripción correspondiente. Tenga en cuenta que la descripción debería modificarse para adecuarse a las circunstancias específicas. Las aplicaciones HTTP no tienen porque gestionar todos los códigos de estado sino los que tenga sentido gestionar en su ámbito. Lo que si deberían controlar sería, al menos, el primer dígito.

"100" Continue	"304" Not Modified	"411" Length Required
"101" Switching Protocols	"305" Use Proxy	"412" Precondition Failed
	"307" Temporary Redirect	"413" Request Entity Too Large
"200" OK		"414" Request-URI Too Large
"201" Created	"400" Bad Request	"415" Unsupported Media Type
"202" Accepted	"401" Unauthorized	"416" Requested range not satisfiable
"203" Non-Authoritative Information	"402" Payment Required	"417" Expectation Failed
"204" No Content	"403" Forbidden	
"205" Reset Content	"404" Not Found	"500" Internal Server Error
"206" Partial Content	"405" Method Not Allowed	"501" Not Implemented
	"406" Not Acceptable	"502" Bad Gateway
"300" Multiple Choices	"407" Proxy Authentication Required	"503" Service Unavailable
"301" Moved Permanently	"408" Request Time-out	"504" Gateway Time-out
"302" Found	"409" Conflict	"505" HTTP Version not supported
"303" See Other	"410" Gone	

Cabeceras de respuesta

Accept-Ranges –

Age – indica el tiempo estimado en segundos desde que se generó la respuesta. Si es 0 posiblemente fue traído del servidor y en caso contrario, será el tiempo que lleva en el proxy cache

Etag –

Location – Sirve para redireccionar a otra localización distinta de la URI existente en la petición

Proxy-Authenticate – Debe indicarse en las respuestas 407 (Proxy Authentication Required), y en su valor estará el esquema y parámetros para autenticarse contra el proxy.

Retry-After – Se suele usar después de un 503 (Service Unavailable) para indicar cuanto tiempo se espera que el recurso no esté disponible. Su valor puede ser una fecha y hora, o bien el número de segundos.

Set-cookie – se usa para enviar cookies desde el servidor al cliente

Set-Cookie: <nombreCookie>=<valorDeLaCookie>; Expires=<fecha>; Max-Age=<nºdeSegundos>; Domain=<dominio>; Path=<ruta>; Secure; HttpOnly

... donde se pueden enviar distintas directivas configurando el tiempo de vida de la cookie (Expires y Max-Age), filtrar los dominios a donde enviarla (Domain), las rutas que deben existir en el servidor antes de enviarla (Path), si enviarla solo si estamos usando HTTPS (Secure), o enviarla de manera que no sean accesibles desde JavaScript en el cliente para evitar XSS (HttpOnly)

Server – Contiene información sobre el software usado por el servidor que gestiona la petición. Se debe usar con precaución, puesto que revelar el software utilizado le hace más vulnerable a sufrir ataques de agujeros de seguridad conocidos para dicho software.

Vary – indica un conjunto de cabeceras de la petición que se pueden cachear para usar en las respuestas para futuras peticiones.

WWW-Authenticate – Debe incluirse en las cabeceras de respuestas 401 Unauthorized. Su valor indicará el esquema de autenticación y sus parámetros (Basic sobre HTTPS, Digest que encripta las credenciales, ...)

En las imágenes existentes a continuación se pueden observar los datos de las cabeceras de las peticiones y respuestas:

- en la navegación web a través de la opción de red (Network) de la herramienta de desarrollo de Firefox – accesible a través de la tecla F12. Más ayuda en https://developer.mozilla.org/es/docs/Tools/Monitor_de_Red
- a través del uso de una extensión (HTTP Request Maker) de Firefox, donde es muy sencillo testear otros tipos de servicios web, y generar nuestras propias cabeceras eligiendo el método, el URI y demás datos complementarios.

Herramientas de desarrollo - EL MUNDO - Diario online líder de información en español - https://www.elmundo.es/

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento

Filtrar las URL

Todos HTML CSS JS XHR Tipografía Imágenes Medios WS Otros Registros pers

Estado	Método	A...	Dominio	Cau
301	GET	/	www.elmun...	docume
200	GET	/	www.elmun...	docume
200	GET	cor...	e00-elmund...	styleshe
200	GET	styl...	e00-elmund...	styleshe
200	GET	ue...	e00-elmund...	script
200	GET	jqu...	e00-elmund...	script
200	GET	ue...	e00-elmund...	script
200	GET	loa...	sdk.privacy-c...	script
200	GET	ch...	static.cha...	script
200	GET	pol...	e00-ue.uecd...	script
200	GET	sat...	assets.ad...	script
200	GET	pix...	pixelcounter....	img
200	GET	cd...	www.elmun...	img
200	GET	ue...	e00-elmund...	script
200	GET	cin...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img

Cabeceras Cookies Parámetros Respuesta Tiempos Tra

URL solicitada: http://www.elmundo.es/
Método de la petición: GET
Dirección remota: 23.60.213.152:80
Código de estado: 301 Moved Permanently ? Editar y volver a enviar
Versión: HTTP/1.1

Filtrar cabeceras

Cabeceras de la respuesta (213 B)

- Connection: keep-alive
- Content-Length: 0
- Date: Sun, 10 Feb 2019 11:08:25 GMT
- Location: https://www.elmundo.es/
- Server: AkamaiGHost
- Vary: User-Agent
- X-UE-Client-Country: ES

Cabeceras de la petición (333 B)

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Encoding: gzip, deflate
- Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
- Connection: keep-alive
- Host: www.elmundo.es
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/64.0

Herramientas de desarrollo - EL MUNDO - Diario online líder de información en español - https://www.elmundo.es/

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento

Filtrar las URL

Todos HTML CSS JS XHR Tipografía Imágenes Medios WS Otros Registros persiste

Estado	Método	A...	Dominio	Cau
301	GET	/	www.elmun...	docume
200	GET	/	www.elmun...	docume
200	GET	cor...	e00-elmund...	styleshe
200	GET	styl...	e00-elmund...	styleshe
200	GET	ue...	e00-elmund...	script
200	GET	jqu...	e00-elmund...	script
200	GET	ue...	e00-elmund...	script
200	GET	loa...	sdk.privacy-c...	script
200	GET	ch...	static.cha...	script
200	GET	pol...	e00-ue.uecd...	script
200	GET	sat...	assets.ad...	script
200	GET	pix...	pixelcounter....	img
200	GET	cd...	www.elmun...	img
200	GET	ue...	e00-elmund...	script
200	GET	cin...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img
200	GET	15...	e00-elmund...	img

Cabeceras Cookies Parámetros Respuesta Tiempos Traz

URL solicitada: https://www.elmundo.es/
Método de la petición: GET
Dirección remota: 23.60.213.152:443
Código de estado: 200 OK ? Editar y volver a enviar Cabeceras sin proces
Versión: HTTP/2.0

Filtrar cabeceras

Cabeceras de la respuesta (390 B)

- cache-control: max-age=4
- content-encoding: gzip
- content-length: 28092
- content-type: text/html; charset=iso-8859-15
- date: Sun, 10 Feb 2019 11:08:25 GMT
- referrer-policy: unsafe-url
- set-cookie: AKA_A2=A; expires=Sun, 10-Feb-...=elmundo.es; secure; HttpOnly
- vary: User-Agent
- vary: Accept-Encoding
- X-Firefox-Spdy: h2
- x-ue-client-country: ES

Cabeceras de la petición (337 B)

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Encoding: gzip, deflate, br
- Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
- Connection: keep-alive
- Host: www.elmundo.es
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/64.0

Herramientas de desarrollo - MARCA - Diario online líder en información deportiva - https://www.marca.com/

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento

Filtrar las URL

Estado	Método	A...	Dominio	Causa
301	GET	/	www.marca....	document
200	GET	/	www.marca....	document
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
200	GET	act...	pagead2....	img

Cabeceras Cookies Parámetros Respuesta Caché Tiempo

URL solicitada: http://www.marca.com/
Método de la petición: GET
Código de estado: 301 Moved Permanently
Versión: HTTP/1.1

Cabeceras de la respuesta (0 B)

- Connection: keep-alive
- Content-Length: 0
- Date: Sat, 09 Feb 2019 10:52:48 GMT
- Location: https://www.marca.com/
- Server: AkamaiGHost
- Vary: User-Agent

Cabeceras de la petición (0 B)

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Encoding: gzip, deflate
- Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
- Connection: keep-alive
- Cookie: ue_privacyPolicy=4; MARCA_idus....marca.com%2F; _cb_svref=null
- Host: www.marca.com
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/64.0

Herramientas de desarrollo - MARCA - Diario online líder en información deportiva - https://www.marca.com/

Inspector Consola Depurador Editor de estilos Rendimiento Memoria Red Almacenamiento

Filtrar las URL

Estado	Método	A...	Dominio	Causa
301	GET	/	www.marca....	document
200	GET	/	www.marca....	document
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
200	GET	act...	pagead2....	img
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
204	POST	csi...	csi.gstatic.com	beacon
200	GET	act...	pagead2....	img
200	GET	pix...	pixelcounter....	img
200	GET	95...	e00-marca.u...	img

Cabeceras Cookies Parámetros Respuesta Tiempos Traz

URL solicitada: https://www.marca.com/
Método de la petición: GET
Dirección remota: 23.60.213.152:443
Código de estado: 200 OK
Versión: HTTP/2.0

Cabeceras de la respuesta (226 B)

- content-encoding: gzip
- content-length: 37001
- content-type: text/html; charset=iso-8859-15
- date: Sun, 10 Feb 2019 10:26:11 GMT
- server: nginx/1.14.0
- vary: Accept-Encoding
- vary: User-Agent
- X-Firefox-Spdy: h2

Cabeceras de la petición (3,152 KB)

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- Accept-Encoding: gzip, deflate, br
- Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
- Connection: keep-alive
- Cookie: ue_privacyPolicy=4; MARCA_idus....marca.com%2F; _cb_svref=null
- Host: www.marca.com
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (Windows NT 10.0; ...) Gecko/20100101 Firefox/64.0

Con la instalación de una extensión de Firefox podremos testear las peticiones y respuestas HTTP

 **Firefox Add-ons** [Explorar](#) [Extensiones](#) [Temas](#) [Más...](#)



HTTP request maker

 by **stefano**

Displays a sidebar that lets you to forge HTTP requests. Press Ctrl+Shift+Y to show the sidebar and make your own HTTP request. You can choose the method GET, HEAD, POST, OPTIONS, PUT DELETE and the body data to send. Response in the main window.

HTTP request Maker

Request

Response

Target Site:

Method:

POST

Request Headers:

Content-Type application/x-www-form-urlencoded
User-Agent Mozilla
Accept */*

Body Data:

nombre=Pepe&apellidos=Perez&nif=12345678Z

Load

Submit

HTTP request Maker

Request

Response

Status Code 200

Response Headers:

connection: Keep-Alive
content-length: 2
content-type: text/html; charset=UTF-8
date: Mon, 11 Feb 2019 23:35:04 GMT
keep-alive: timeout=5, max=100
server: Apache/2.4.29 (Win32) OpenSSL/1.1.0g
PHP/7.2.2
x-powered-by: PHP/7.2.2

Response Data:

11

Servicios web

Los servicios web son componentes distribuidos de una aplicación, accesibles externamente, generalmente usados para acceder desde aplicaciones escritas en distintos lenguajes y ejecutadas desde distintas plataformas. Esta independencia se consigue gracias al uso de estándares como XML y HTTP.

Son alternativa a otras arquitecturas de computación distribuida como son CORBA (Common Object Request Broker Architecture), RMI (Java Remote Method Invocation) o DCOM (Distributed Component Object Model). Éstas arquitecturas son más eficaces al no estar basadas en texto como los servicios web, y más preparadas para la realización de operaciones comunes como las transacciones. Pero la independencia de la plataforma y la flexibilidad de los servicios web suplen estas carencias.

Existen dos tipos de implementación de los servicios web: SOAP y REST

SOAP (Simple Object Access Protocol)

Son los servicios web tradicionales, y se basan en la exposición de sus interfaces en documentos públicos en un formato XML denominado WSDL (Lenguaje de descripción de servicios web, Web Services Description Language). En dicho documento se incluye por ejemplo la localización del servicio web y el interfaz de sus posibles operaciones (los métodos que podemos invocar, así como sus parámetros, tipos de datos, etc). Esta información sirve para saber cómo utilizarlo, pero no qué es lo que hace. Un archivo WSDL tiene un esquema como el siguiente:

```
<definitions>

<types> definición de los tipos de datos, al estilo de los esquemas XML (XSD)
  <schema ...>
    <element name='nombreElemento'> ... </element>
  </schema>
</types>

<message name='nombreMensaje1'> definición de los elementos de un mensaje
  <part name='body' element='nombreElemento'>
</message>

<message name='nombreMensaje2'> ... </message>
<message name='nombreMensaje3'> ... </message>

<portType name='nombreOperaciones'> definimos las operaciones existentes y mensajes intercambiados
  <operation name='nombreOperacion'>
    <input message='nombreMensaje1' />
    <output message='nombreMensaje2' />
  </operation>
</portType>

<binding name='nombreBinding'> definimos protocolos de comunicación utilizados en cada operación
</binding>
<service> definimos el nombre y direcciones del servicio web creado
  <documentation>Mi super servicio web</documentation>
  <port name='nombreOperaciones' binding='nombreBinding'>
</service>

</definitions>
```

El cliente se basará en dicha información para realizar la comunicación posterior, que se hará a través de mensajes en un formato XML denominado SOAP. Es un buen mecanismo para grandes aplicaciones con comunicaciones complicadas y con requerimientos especiales de seguridad.

Un archivo SOAP tiene un esquema como el siguiente:

```
<?xml version="1.0"?>
<soap:Envelope      xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
                    Soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>

  </soap:Header>
  <soap:Body>
  ...

  <soap:Fault>      Utilizado para indicar mensajes de error
  </soap:Fault>

  </soap:Body>
</soap:Envelope>
```

Una comunicación con SOAP podría ser la siguiente, donde se produce una primera petición aportando una clave de una empresa de bolsa ...

```
<?xml version="1.0"?>

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

  <soap:Body>
    <cdm:GetCotizacion xmlns:cdm="https://www.ieschandomonte.edu.es/webservices/cotizacion">
      <cdm:Empresa>ZELTIA</cdm:Empresa>
    </cdm:GetCotizacion>
  </soap:Body>

</soap:Envelope>
```

y el servicio web responde con datos sobre su cotización ...

```
<?xml version="1.0"?>

<soap:Envelope
  xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
  soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

  <soap:Body>
    <cdm:GetCotizacionResponse xmlns:cdm="https://www.ieschandomonte.edu.es/webservices/cotizacion">
      <cdm:CotizacionActual>7.11</cdm:CotizacionActual>
      <cdm:CotizacionUltimoCierre>6.66</cdm:CotizacionUltimoCierre>
    </cdm:GetCotizacionResponse>
  </soap:Body>

</soap:Envelope>
```

REST (Representational State Transfer)

Roy Fielding (uno de los autores de la especificación HTTP) presentó en el año 2000 un estilo de arquitectura para sistemas distribuidos a través del protocolo HTTP. Cualquier arquitectura que cumpla con una serie de requisitos, se puede considerar REST (RESTful se suelen denominar a los servicios web que cumplen estas restricciones).

- Separación de cliente y servidor: sus implementaciones se realizan independientemente y uno no conoce los detalles del otro. Simplemente los distintos clientes realizan peticiones al servidor, y reciben una respuesta en un formato que ambos conocen (XML, JSON, ...)
- Sin estado (*stateless*): cada petición es independiente y lleva toda la información necesaria, por lo que no es necesario mantener sesiones (el servidor no almacenará información, toda estará en el cliente).
- Cacheable: Las respuestas del servidor llevarán información acerca de si sus datos son cacheables o no, lo cual evitaría repetir conexiones entre cliente y servidor para recuperar un mismo recurso.
- Interfaz uniforme: existirá una interfaz genérica para administrar las interacciones entre el cliente y el servidor. Para ello existirá una manera única de **identificar un recurso (a través de una URI)** y una serie de **operaciones básicas aplicables (métodos o verbos HTTP)**. Mediante cabeceras HTTP como Accept o Content-Type se puede especificar la **representación** que nos interesa para transportar el recurso.
- Sistema de capas: el servidor se implementará con una arquitectura en capas donde cada componente está construido sobre su capa anterior, mejorando la escalabilidad y la seguridad.

Es una nueva manera de comunicación basado en modificar las operaciones de las cabeceras HTTP tratando los recursos como URIs, y especificando el método o verbo adecuado.

Los verbos que se suelen utilizar son los asociados a las operaciones básicas CRUD:

- | | | |
|----------|--------|---------------------------------|
| - GET | Read | Leer datos del servidor |
| - POST | Create | Crear datos en el servidor |
| - PUT | Update | Actualizar datos en el servidor |
| - DELETE | Delete | Eliminar objetos del servidor |

De la ruta especificada en la URL obtenemos el recurso sobre el que actúa el verbo.

Rutas adecuadas pueden ser las siguientes (utilizando sustantivos, nunca verbos)

- ~~/clientes.php/~~ Devuelve (GET) todos los clientes
- ~~/clientes.php/~~cliente Inserta (POST) un cliente
- ~~/clientes.php/~~cliente/{codCliente} Devuelve(GET),actualiza(PUT) o elimina(DELETE) el codCliente
- ~~/clientes.php/~~provincia/{codProvincia} Devuelve (GET) los clientes de la provincia indicada.

NOTA: Modificar el archivo .htaccess para que se puede llamar al servicio web rest sin especificar el nombre del archivo PHP

REST vs SOAP	REST	SOAP
Significado	Representational State Transfer	Simple Object Access Protocol
Diseño	Estilo de arquitectura con guías y recomendaciones	Protocolo estándar con sus reglas bien definidas
Estilo de uso	Orientado a datos: /clientes	Orientado a funciones: getClientes()
Complejidad	Sencillo	Algo más complejo
Rendimiento	Muy ligero	Pesado, necesita más ancho de banda
Formato de los datos	JSON,XML,YAML,CSV,RSS, etc.	XML
Sin estado	Sí	Sí, pero es fácil mantener el estado
Protocolos para transferir	Sólo HTTP	HTTP, SMTP, UDP, etc.
Métodos usados	GET,POST,PUT,DELETE	Suele usar POST
Seguridad	No (sí con HTTPS / SSL)	Sí
Cacheable	Sí	No
Usos	APIs públicas de servicios móviles, redes sociales, etc.	Apps empresariales, seguras, pasarelas de pago, telecomunicaciones, etc.

Implementación de un servicio web REST en PHP

Para implementar un servicio web REST es necesario recoger, al menos, los dos siguientes datos:

`$_SERVER['REQUEST_METHOD']` es el verbo indicado por el cliente

`$_SERVER['PATH_INFO']` es la ruta que indica el recurso sobre el que actuaremos

```
$verbo=$_SERVER['REQUEST_METHOD'];
$pathInfo=isset($_SERVER['PATH_INFO'])?trim($_SERVER['PATH_INFO'],'/'):'';
$ rutas=$pathInfo=='?'?:explode('/', $pathInfo);

array_walk($rutas,function ($elemento) { echo "$elemento<br>"; });
```

De ahí obtendremos los datos necesarios para interpretar la ruta especificada al servicio, y el verbo de acción que queramos aplicarle.

Si se necesitan más datos (POST O PUT) se obtendrán del flujo de entrada `php://input`

```
parse_str(file_get_contents('php://input', true), $datos);
```

La llamada a `parse_str` lee el texto devuelto por el `file_get_contents` sobre el flujo de entrada y, asumiendo su sintaxis tipo string de consulta de una URL (`var1=valor1&var2=valor2&...`), crea dichas entradas en el array asociativo pasado como segundo parámetro (`$datos`).

El servicio web REST devolverá lo siguiente:

- Código de respuesta HTTP adecuado al resultado de la acción solicitada.

Se utilizará para ello la función `http_response_code ($codigo);`

200	OK	Operación correcta
201	Created	Recurso creado (INSERT)
204	No Content	Operación correcta que no devuelve ningún contenido (DELETE)
301	Moved Permanently	El recurso se ha movido a la URI indicada
304	No Modified	El recurso no ha cambiado
401	Unauthorized	No está autorizado a acceder al recurso
404	Not Found	El recurso especificado no existe
405	Method Not Allowed	Método no permitido
422	Unprocessable Entity	Faltan parámetros o son incorrectos
429	Too Many Requests	Si hay un número limitado de peticiones, el usuario las ha superado
500	Internal Server Error	Existe un error en el servidor
503	Service Unavailable	Servidor temporalmente fuera de servicio

- Opcionalmente un contenido

Estos datos devueltos por el servicio web REST al cliente (el/los recursos pedidos (GET), el código del recurso insertado (POST), etc.) suelen estar codificados en formato JSON (ver anexo), aunque se podrían utilizar otros formatos similares como YAML o XML.

Estos resultados serán directamente la salida del script PHP (generado con *echo* o con contenido estático). Es responsabilidad del cliente analizarlos adecuadamente.

Probar el servicio web

- Escribiendo nuestro propio cliente realizando las llamadas HTTP precisas indicando el método y la ruta adecuada, y parseando el resultado devuelto en función del formato utilizado.
- Utilizando utilidades genéricas que nos permiten probar las funcionalidades de un api REST. Existen diversas aplicaciones (p.e. Insomnia), varias extensiones del navegador Chrome (p.e. Advanced REST client), así como herramientas integradas en los distintos *frameworks*.
- Desde un navegador probando por GET y por POST desde un formulario HTML. Con la extensión adecuada (u opciones del propio navegador, p.e. Desarrollador Web del Firefox) existen más posibilidades.


```
<?php

/* Rutas permitidas
(si utilizamos redirecciones en .htaccess, eliminar el clientes.php)

/clientes.php                               Devuelve (GET) todos los clientes
/clientes.php/cliente                       Inserta (POST) un cliente
/clientes.php/cliente/{codCliente}         Devuelve (GET), actualiza (PUT) o elimina (DELETE) codCliente
/clientes.php/provincia/{codProvincia}     Devuelve los clientes de codProvincia
*/

class HTTP {
    const resultados = array(
        'OK'=>200,
        'CREATED'=>201,
        'NO CONTENT'=>204,
        'NO MODIFIED'=>304,
        'BAD REQUEST'=>400,
        'UNAUTHORIZED'=>401,
        'NOT FOUND'=>404,
        'METHOD NOT ALLOWED'=>405,
        'UNPROCESSABLE ENTITY'=>422,
        'TOO MANY REQUEST'=>429,
        'INTERNAL SERVER ERROR'=>500,
        'SERVICE UNAVAILABLE'=>503
    );

    public static function resultado($error) {
        if(!array_key_exists($error, HTTP::resultados))
            $error='BAD REQUEST';
        http_response_code(HTTP::resultados[$error]);
        exit();
    }
}

function getDatos() {
    parse_str(file_get_contents('php://input', true), $datos);
    return $datos;
}

function json_encode_cursor_db($filas) {
    $json = array();
    while($fila=$filas->fetch_assoc())
        $json[]=$fila;
    switch(count($json)) {
        case 0: return false;
        case 1: return json_encode($json[0]);
        default: return json_encode($json);
    }
}
```

```
// Creamos la conexión con la Base de Datos
$servername = "localhost";
$username = "root";
$password = "";
$db = 'clientes';
$con = new mysqli($servername, $username, $password, $db);
if ($con->connect_error)
    die("Error en la conexión: " . $con->connect_error);
$con->set_charset('UTF8');

$verbo=$_SERVER['REQUEST_METHOD'];
$pathInfo=isset($_SERVER['PATH_INFO'])?trim($_SERVER['PATH_INFO'],'/'):'';
$ruta=$pathInfo=='?'?:explode('/', $pathInfo);

if(count($ruta)==0) {
    if($verbo=='GET') {
        $sql='SELECT * FROM Clientes ORDER BY apellidos,nombre';
        $json=json_encode_cursor_db($con->query($sql));
        if($json) {
            echo $json;
            HTTP::resultado('OK');
        }
        else
            HTTP::resultado('NOT FOUND');
    }
    else
        HTTP::resultado('METHOD NOT ALLOWED');
}

elseif(count($ruta)==1) {
    if($ruta[0]=='cliente') {
        if($verbo=='POST') {
            $datos=getDatos();
            $nombre=$datos['nombre'];
            $apellidos=$datos['apellidos'];
            $nif=$datos['nif'];
            $sql="INSERT INTO Clientes (nombre,apellidos,nif) " .
                "VALUES ('$nombre','$apellidos','$nif)";
            $con->query($sql);
            if($con->affected_rows==1) {
                echo json_encode(array('id'=>$con->insert_id));
                HTTP::resultado('CREATED');
            }
            else
                HTTP::resultado('INTERNAL SERVER ERROR');
        }
        else
            HTTP::resultado('METHOD NOT ALLOWED');
    }
    else
        HTTP::resultado('UNPROCESSABLE ENTITY');
}
}
```

```

elseif (count($ruta)==2) {
    $parametro=$ruta[1];
    if(!preg_match('/^\d+$/',$parametro))
        HTTP::resultado('UNPROCESSABLE ENTITY');
    switch($ruta[0]) {
        case 'cliente':
            switch($verbo) {
                case 'GET':
                    $sql="SELECT * FROM Clientes WHERE codCliente=$parametro";
                    $json=json_encode_cursor_db($con->query($sql));
                    if($json) {
                        echo $json;
                        HTTP::resultado('OK');
                    }
                    else
                        HTTP::resultado('NOT FOUND');
                case 'PUT':
                    $datos=getDatos();
                    $nombre=$datos['nombre'];
                    $apellidos=$datos['apellidos'];
                    $nif=$datos['nif'];
                    $sql = "UPDATE Clientes SET nombre='$nombre',
                        apellidos='$apellidos', nif='$nif'
                        WHERE codCliente=$parametro";
                    $con->query($sql);
                    if($con->affected_rows==1)
                        HTTP::resultado('NO CONTENT');
                    else
                        HTTP::resultado('NOT FOUND');
                case 'DELETE':
                    $sql="DELETE FROM Clientes WHERE codCliente=$parametro";
                    $con->query($sql);
                    if($con->affected_rows==1)
                        HTTP::resultado('NO CONTENT');
                    else
                        HTTP::resultado('NOT FOUND');
                default:
                    HTTP::resultado('METHOD NOT ALLOWED');
            }
        case 'provincia':
            if($verbo=='GET') {
                $sql="SELECT * FROM Clientes WHERE codProvincia=$parametro
                    ORDER BY apellidos,nombre";
                $json=json_encode_cursor_db($con->query($sql));
                if($json) {
                    echo $json;
                    HTTP::resultado('OK');
                }
                else
                    HTTP::resultado('NOT FOUND');
            }
            else
                HTTP::resultado('METHOD NOT ALLOWED');
        default:
            HTTP::resultado('UNPROCESSABLE ENTITY');
    }
}
else
    HTTP::resultado('NOT FOUND');

```

Aspectos de seguridad

Mensajes de error

Para tener un código seguro es fundamental gestionar de manera adecuada los errores o avisos que PHP nos proporciona sobre nuestro código.

Por ejemplo, con la directiva *display_errors Off* evitamos que dichos errores se vean en la página, lo cual podría suponer un problema de seguridad (además de quedar muy feo). Esto es prácticamente obligatorio cuando la web está en producción, no así en desarrollo.

La directiva *error_reporting* (o la función `error_reporting()`), se establece con un nivel de notificación de errores. Este nivel de notificación se calcula en función de unas [constantes predefinidas](#) (`E_ERROR`, `E_WARNING`, `E_PARSE`, `E_`, `E_DEPRECATED`, `E_ALL`, ...).

Una posibilidad interesante es deshabilitar los errores en pantalla (como ya hemos comentado) y hacer que se almacenen en un archivo. Para ello activaremos la directiva *log_errors On* y *error_log* a la ruta del archivo de log donde queramos almacenar dicha salida.

Register Globals (obsoleta/eliminada)

Desactivar la directiva *register_globals* del `php.ini` para evitar la creación de variables globales de manera masiva. Si desarrollamos una aplicación en un entorno en donde esté activado, es importante inicializar todas las variables y establecer *error_reporting* a `E_ALL | E_STRICT` para que se nos avise del uso de variables sin inicializar. Esta característica en PHP 4 estaba activada por defecto, en 4.2 desactivada por defecto, en 5.3 se declaró como obsoleta y **a partir de 5.4 ya ha sido eliminada**.

Magic Quotes (obsoleta/eliminada)

Activar la directiva *magic_quotes_gpc* permite limpiar automáticamente los datos de entrada a un script PHP escapando caracteres potencialmente peligrosos como comillas, barras, etc. Pero esta directiva también ha sido declarada obsoleta en PHP 5.3 y **eliminada en PHP 5.4** por problemas de rendimiento y por modificar innecesariamente, en algún caso, los datos.

Re-autenticaciones

Incluso en una aplicación sin aparentes fallos de seguridad, no es mala idea pedir la autenticación de nuevo antes de realizar operaciones potencialmente delicadas. Esto evitará, por ejemplo, que alguien realice algo fraudulento en nuestro terminal aprovechando un descuido, una pequeña ausencia, o recuperar la navegación en una sesión que no fue cerrada correctamente.

Este comportamiento es típico en webs bancarias en donde, una vez autenticados, nos pregunta de nuevo antes de realizar movimientos en alguna cuenta.

Hash de contraseñas seguro

En este apartado comentaremos los aspectos relevantes en el almacenamiento de contraseñas de usuario. Ha sido extraído de <https://www.php.net/manual/es/faq.passwords.php>

¿Por qué debo usar hash en las contraseñas de los usuarios de mi aplicación?

El hash de contraseñas es una de las consideraciones de seguridad más elementales que se deben llevar a la práctica al diseñar una aplicación que acepte contraseñas de los usuarios. Sin hashing, cualquier contraseña que se almacene en la base de datos de la aplicación podrá ser robada si la base de datos se ve comprometida, con lo que inmediatamente no sólo estaría comprometida la aplicación, sino también las cuentas de otros servicios de nuestros usuarios, siempre y cuando no utilicen contraseñas distintas.

Si aplicamos un algoritmo hash a las contraseñas antes de almacenarlas en la base de datos, dificultamos al atacante el determinar la contraseña original, pese a que en un futuro podrá comparar el hash resultante con la contraseña original.

Sin embargo, es importante tener en cuenta que el hecho de aplicar hash a las contraseñas sólo protege de que se vean comprometidas las contraseñas almacenadas, pero no las protege necesariamente de ser interceptadas por un código malicioso inyectado en la propia aplicación.

¿Por qué las funciones hash como md5() y sha1() no son adecuadas para las contraseñas?

Los algoritmos hash como MD5, SHA1 o SHA256 están diseñados para ser muy rápidos y eficientes. Con las técnicas y equipos modernos, es algo trivial extraer por fuerza bruta la salida de estos algoritmos, para determinar los datos de entrada originales.

Dada la velocidad con que los ordenadores actuales pueden "invertir" estos algoritmos hash, muchos profesionales de la seguridad recomiendan encarecidamente no utilizarlas como funciones hash para contraseñas.

¿Qué hash debo aplicar a mis contraseñas, si las funciones hash más comunes no son adecuadas?

Al aplicar un algoritmo hash, **los dos factores más importantes son el coste computacional y la sal**. Cuanto más cueste aplicar un algoritmo hash, más costará analizar su salida por fuerza bruta.

PHP 5.5 proporciona una API de hash de contraseñas nativa que maneja cuidadosamente el empleo de hash y la verificación de contraseñas de una manera segura.

¿Qué es una sal (salt)?

Una sal criptográfica es un dato que se utiliza durante el proceso de hash para eliminar la posibilidad de que el resultado pueda buscarse a partir de una lista de pares precalculados de hash y sus entradas originales, conocidas como **tablas rainbow**.

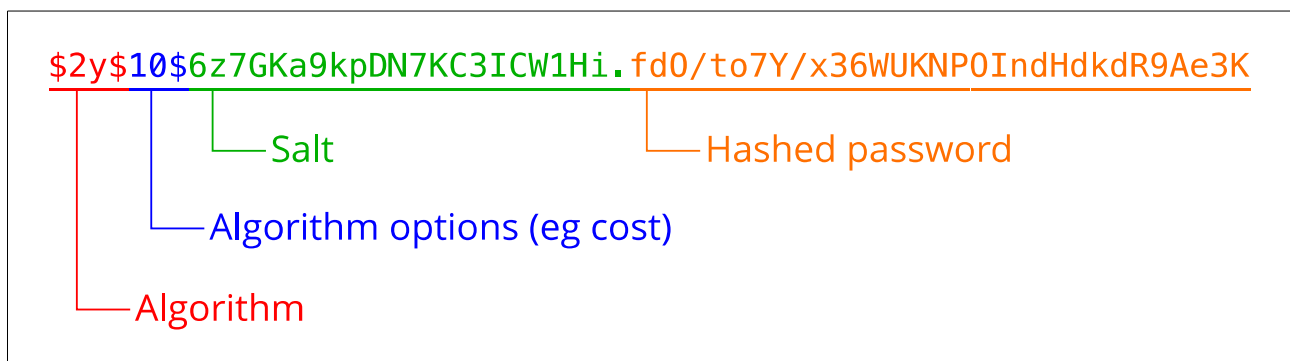
Es decir, una sal es un pequeño dato añadido que hace que los hash sean significativamente más difíciles de crackear. Existe un gran número de servicios online que ofrecen grandes listas de códigos hash precalculados, junto con sus datos de entrada originales. El uso de una sal hace muy difícil o imposible encontrar el hash resultante en cualquiera de estas listas.

password_hash() creará una sal aleatoria si no se proporciona una, siendo esta generalmente la estrategia más sencilla y segura.

¿Cómo almaceno mis sales?

Al utilizar **password_hash()**, el valor devuelto incluye la sal como parte del hash generado. Este valor debería almacenarse tal cual en la base de datos, ya que incluye información sobre la función hash que se empleó y así proporcionarla directamente a **password_verify()** al verificar contraseñas.

El siguiente diagrama muestra el formato de un valor devuelto por **password_hash()**. Como se puede observar, son autocontenidos, con toda la información del algoritmo y la sal requerida para futuras verificaciones de contraseñas.



Plantillas

La necesidad de separar el interfaz generado (HTML) con la lógica de la generación (PHP), nos hace plantearnos la necesidad de utilizar algún sistema de plantillas.

Algunos de los más utilizados son:

- Mustache: sus características más importante son su soporte multilenguaje y que es 'logic-less', es decir, carece de estructuras explícitas alternativas o repetitivas.



- Smarty: programado y pensado en y para PHP, permite introducir código por lo que pierde independencia, pues puede parecer que es un subnivel de PHP dentro de la plantilla.



- Twig: motor de plantillas del Symfony Framework, soportado su comunidad.

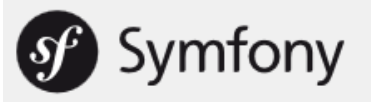




- Blade: motor de plantillas del Laravel Framework

Frameworks PHP:

Existen distintos frameworks para realizar aplicaciones complejas en PHP, que se compone de un conjunto de herramientas que permiten programar más rápido y construir aplicaciones más robustas. Para ello suelen basar su modelo de desarrollo en una arquitectura de capas Modelo Vista Controlador (MVC).

A continuación se muestran algunos muy conocidos, junto con el ORM (mapeador objeto-relacional) y el motor de plantillas para construir las vistas que suelen utilizar por defecto.

Frameworks	ORM	Plantillas	Línea de comandos
	Doctrine, Propel	Twig	php symfony
	Eloquent	Blade+PHP	php artisan
	Yii Active Record	Twig, Smarty, ...	php yii

Lenguaje Javascript

Con Javascript se pueden desarrollar de manera sencilla páginas web interactivas pues permite plena interacción con la página y el navegador. De hecho, el navegador y las propias etiquetas que componen un documento HTML se “reflejan” en una serie de objetos cuyas propiedades pueden cambiarse o manipularse desde el lenguaje de script.

Algunos de los típicos usos de Javascript son:

- Generación HTML dinámica (“al vuelo”), en (y desde) la página local.
- Validación de campos de un formulario HTML antes de ser enviados a un servidor
- Obtención de entrada local del usuario con el fin de permitir que el usuario tenga oportunidad de llamar a opciones distintas dentro del navegador
- Mostrar al usuario mensajes dentro de ventanas
- Manipulación y asignación del foco
- Detección e interacción entre applets de Java y aplicaciones auxiliares (plugins)
- Uso de AJAX
- Gestionar cookies
- etc.

El lenguaje Javascript posee las siguientes características generales:

- Su sintaxis (reducida) proviene del lenguaje C.
- Las diferentes instrucciones se separan por un punto y coma
- Se distingue entre mayúsculas y minúsculas (es *case-sensitive*)
- Las variables se declaran con: *var nombreVar [=valor];*
- El tipo no se define cuando se crea una variable sino que se cambia de tipo en función del valor que tiene. Los tipos internos que maneja son: *undefined*, *boolean*, *number*, *string*, *object* y *null*.
- Las funciones se crean con la palabra clave *function* y no se especifica su valor de retorno (si existe), simplemente se devolverá con *return*.

Localización del código Javascript

El código javascript puede aparecer casi en cualquier parte del documento, y va siempre embebido en el código HTML dentro de las etiquetas `<SCRIPT> ... </SCRIPT>`, con la única excepción del código Javascript asociado a un manejador de evento de un elemento HTML.

En este último caso, el código (*) se situará directamente como valor del atributo existente que represente al evento que queremos controlar:

```
<body onload='*'>
<form onsubmit='*'>
<a href='...' onclick='*'>
<img onmouseover='*' onmouseout='*'>
<input type='text' onkeydown='*'>
<select onchange='*'>
```

En el caso de querer situar el código Javascript en un archivo externo (por diseño o por reutilización), la etiqueta `<SCRIPT>` dispone de un atributo `src` cuyo valor sería la URL donde reside un archivo de texto que contiene el script (en este caso el contenido de `<SCRIPT>` estaría vacío).

Para asegurar que las funciones Javascript se cargan en memoria tan pronto el navegador lee el documento, el código se suele colocar en el interior del elemento `<HEAD>`.

En todo caso, los scripts que generan HTML en partes específicas de un documento, deben colocarse donde sea necesario.

Expresiones regulares en Javascript

No es objetivo de este manual valorar la importancia del dominio de las expresiones regulares en cualquier lenguaje de programación. Javascript no es una excepción. Bien sea con los valores de los controles de un formulario, o con el texto de un archivo, las expresiones regulares pueden ahorrarnos muchas líneas de código y evitarnos muchos errores (ver anexo).

```
var expresionregular = /patrónER/modificadores;
```

El patrónER consiste en el conjunto de caracteres (normales y especiales) de las expresiones regulares estándar que define el patrón con el que se intentará encajar el texto sobre el que trabajemos.

Los modificadores, si se incluyen, será alguno de los siguientes:

- **i** encaja ignorando mayúsculas-minúsculas
- **g** busca todas las coincidencias, no para al encontrar la primera
- **m** realiza una búsqueda en textos multilínea

Muchas funciones de cadenas permiten utilizar como parámetros expresiones regulares, como por ejemplo *search()* y *replace()*. Pero la función que directamente nos informa de si una cadena encaja (o no) con una expresión regular es *test()*.

```
<script>
    var er = /\d+/;
    document.write(er.test("5X"));
    // Visualiza true
</script>

<script>
    var er = /^\d+$/;
    document.write(er.test("5X"));
    // Visualiza false
</script>
```

Acceso a los elementos de la página

Para acceder a los elementos de la página utilizamos el HTML DOM (Document Object Model) a través del objeto *document*.

Existen multitud de métodos y propiedades para acceder, añadir, eliminar, etc. a todos los elementos, atributos y eventos. Ver en http://www.w3schools.com/js/js_html_dom_document.asp

Para acceder a un elemento directamente utilizaremos *document.getElementById('id');*

```
<script>
function datosOk() {
    var cajaTextoNif = document.getElementById("idNif");
    var valorNif = cajaTextoNif.value;
    ...
}
</script>

<form action='consultar.php' method='POST' name='f' onSubmit='return datosOk()'>
    <input type='text' name='nif' id='idNif'>
    ...
</form>
```

También podríamos acceder a través de su atributo *name* utilizando el árbol DOM.

```
<script>
function datosOk() {
    var cajaTextoNif = document.forms.f.nif;
    var valorNif = cajaTextoNif.value;
    ...
}
</script>
```

Acceso a las propiedades de elementos de la página

Una vez accedemos a un elemento, podremos acceder a sus propiedades que, por lo general, son un reflejo de sus atributos.

Suele ser habitual acceder a:

- la propiedad *value* de una caja de texto
- la propiedad *src* de una imagen
- al array *options* de una lista, a sus propiedad *selectedIndex* o al *value* de sus elementos
- la propiedad *innerHTML* de un párrafo `<p>` o un ``

El modelo de objetos

window: hace referencia a la ventana principal del navegador

Propiedades:

- frames: array que contiene los marcos del documento (definidos con frameset)
- length: número de marcos del documento
- name: título de la ventana actual, tal como se pasó en el método open
- status: mensaje en la barra de estado del navegador

Métodos:

- alert: desplegar un mensaje de texto
- confirm: permite introducir opciones sí/no y devuelve un valor booleano
- prompt: solicita información (textual) al usuario
- open: abre una ventana
- close: cierra la ventana
- setTimeout: agrega un timer a una ventana
- clearTimeout: elimina el timer

document: es un reflejo del cuerpo del documento (`<body>...</body>`). Proporciona los métodos para acceder a los elementos de la página, así como para generar dinámicamente el HTML.

Propiedades:

- anchors: array que refleja las marcas ``
- bgColor: color de fondo de la etiqueta `<body>`
- fgColor: atributo text de la etiqueta `<body>`
- forms: array con los formularios del documento
- images: array con las imágenes `` del documento
- lastModified: última fecha en que se modificó
- linkColor: refleja el color de los enlaces
- links: array con los enlaces `` del documento
- location: URL del documento actual
- referrer: refleja el URL del documento que hizo la llamada al documento
- title: título del documento
- vlinkColor: color del enlace visitado

Métodos:

- getElementById("id"): devuelve el elemento con el id indicado
- getElementsByTagName("tag"): devuelve un array con los elementos con esa etiqueta (p, li, etc.)
- clear: limpia el documento actual
- write, writeln: escribir cadenas de texto HTML en el punto actual

location: reflejo de la URL del documento**Propiedades:**

- host: especifica la parte nombreServidor:puerto de la URL actual
- hostname: especifica el nombre completo del servidor (incluyendo su dominio o IP)
- href: especifica el URL completo del documento
- pathname: especifica la trayectoria de la URL(después del nombre de servidor)
- port: especifica el puerto de comunicaciones que usa el servidor
- protocol: especifica el protocolo utilizado (incluido el signo de dos puntos)

history: es una lista de los URL (sitios o páginas) visitados en la sesión en curso y que están contenidos en el menú Ir del navegador. Un conjunto de métodos asociados al objeto history permiten que se carguen distintos URLs en el navegador y que se pueda navegar hacia adelante y atrás entre los URLs previamente cargados.

Propiedades:

- length: número de entradas en el historial

Métodos:

- back: carga el URL anterior
- forward: carga el URL posterior
- go(nº): va al número de página relativo especificado

navigator: Contiene detalles del navegador instalado.

Propiedades:

- appName: nombre del navegador
- appVersion: versión del navegador
- plugins: array de los plug-ins instalados en el sistema
- mimeTypes: array de tipos MIME que soporta.

MIME (Multipurpose Internet Mail Extensions) es un modo estándar que categoriza distintos formatos de archivo. Por ejemplo, una imagen GIF se representa mediante el nombre image/gif, mientras que un archivo HTML se representa con el nombre text/html y así sucesivamente. Cada aplicación auxiliar que se instala actualiza el navegador para que reconozca el nuevo tipo MIME.

Validación de formularios en el cliente

La validación de formularios desde Javascript tiene la función principal de avisar al usuario de posibles errores en la introducción de datos antes de que estos viajen al servidor. Con esto conseguimos una mejor respuesta en el interfaz de la aplicación, pues los errores se identifican en el mismo momento de su introducción, además de minimizar la sobrecarga del servidor al no perder el tiempo validando datos erróneos.

De todos modos hay que entender dos cosas:

- No todos los datos se pueden validar en el cliente: por ejemplo, aquellas validaciones que involucren consultas a la base de datos residente en el servidor. Cuando vamos a darnos de alta en un sistema que nos permite elegir nuestro identificador, que debe ser único, el sistema no puede comprobar su unicidad sin comunicarse con el servidor en algún momento.
- Las validaciones en el lado del cliente son únicamente para que nuestra web funcione de manera más ágil, pero nunca deben reemplazar a la necesaria validación en el servidor. Esto es debido a que, entre otros motivos, el servidor no puede saber seguro de dónde vienen los datos. Alguien puede modificar los parámetros enviados (independientemente de si van por GET por POST), o directamente enviar los datos sin pasar por los filtros del cliente establecidos. En definitiva, el código situado en el servidor debe validar todo lo necesario para asegurar la integridad de los datos.

Capturar los eventos necesarios

Lo primero que hay que hacer a la hora de validar las acciones de los usuarios en la página web es preguntarse cuándo queremos controlar que dichas acciones han sido correctas (que los datos introducidos son válidos, por ejemplo).

Lo normal es esperar a que el usuario indique, con la pulsación de un botón (*submit*), que se envíen los datos al servidor. En ese momento podremos hacer un repaso de todos los datos para verificar que, en la medida de lo posible, está todo correcto.

Esta situación se puede controlar de varias maneras:

- Controlando el evento *onClick* de un botón de tipo *submit*. Este tipo de botones tiene una acción predeterminada que es el envío del formulario en el cual se encuentran, por lo que si no hacemos nada, siempre se enviarán los datos. En el caso de que el usuario haya pulsado el botón y no queramos que se envíen los datos (normalmente porque hay algún error en ellos), únicamente tendremos que ejecutar el código javascript *return false*; lo cual 'cancela' el evento como si no hubiera ocurrido (y no se enviarán los datos del formulario).
- Controlando el evento *onClick* de un botón (tipo *button*). Como este botón no tiene un comportamiento por defecto asociado, aquí lo que habrá que hacer es forzar el envío del formulario (si nos interesa). Esto se hará accediendo al formulario y llamando a su método *submit()*.
- Controlando directamente el evento *onSubmit()* del propio formulario – recomendado – . De esta manera, independientemente de la acción que desencadene el envío del formulario, siempre se controlará su acción capturando este evento.

De todas maneras, no siempre es necesario esperar a que el usuario indique que quiere enviar el formulario para realizar toda la validación. A veces nos puede interesar controlar acciones intermedias en la web para agilizar el proceso y no esperar al final. Por ejemplo, si se cubre mal un código postal o una fecha, podemos detectarlo inmediatamente y aprovechar que dicho control ya posee el foco para que se corrija en el momento.

Otras veces es casi obligatorio controlar acciones intermedias, sobre todo cuando el comportamiento de la página depende de un dato introducido. Por ejemplo, si desplegamos una lista de provincias, queremos que la siguiente lista muestre una lista de los ayuntamientos de la provincia seleccionada. Para estas situaciones capturar el evento *onChange* es lo habitual.

Detección de los errores

Básicamente se reduce a acceder a los controles adecuados (explicado en un epígrafe anterior) y verificar que el dato existente, introducido por el usuario, es correcto. Existen multitud de posibilidades, pero algunas de las más comunes son:

- Verificar únicamente la existencia de algún valor: p.e. se requiere introducir un nombre. Habitualmente se reduce a acceder a la propiedad *value* de una caja de texto y verificar si contiene algún valor. De todas formas, esta validación se puede hacer directamente desde HTML añadiendo el atributo *required=""* (y opcionalmente el evento *onInvalid*) , evitando así el código javascript, aunque el error no es tan personalizado y no se controlan los espacios en blanco.

```
<!DOCTYPE html>
<html>
<head>
  <title>Validaciones</title>
  <script>
    function formOk() {

      var cajaNombre = document.getElementById("id");
      var nombre = cajaNombre.value.trim();
      if(nombre=="") {
        alert("No se ha introducido el nombre.");
        cajaNombre.value=""; // por si había espacios
        cajaNombre.focus();
        return false;
      }
      // otras validaciones ...
      return true;
    }
  </script>
</head>
<body>
<form onSubmit='return formOk();'>
  Nombre <input type='text' id='id'>
  <input type='submit' value='Probar'>
</form>
</body>
</html>
```

- Verificar el rango de un valor numérico: muy similar al anterior, además debemos controlar algún intervalo numérico desde Javascript. Aquí lo importante es que antes de verificar dicho rango, debemos cerciorarnos de que el contenido del control es efectivamente un número, porque en caso contrario podríamos tener problemas a la hora de comparar. Aquí de nuevo se podrían utilizar nuevos atributos HTML para delegar en el navegador y olvidarnos del código. A continuación puede observar el comportamiento curioso del mensaje de error del siguiente HTML, teniendo en cuenta que 5 sí es un valor válido.

```
<input type='number' id='id' required=''' min='5' max='10'>
```

Cantidad

Seleccione un valor que sea mayor que 5.

- Los nuevos tipos de controles HTML nos dan una gran potencia a la hora de filtrar la entrada de datos del usuario y por tanto, evitar validaciones desde código. Entre esos tipos estarían *number*, *range*, *date*, *datetime*, *month*, *week*, *color*, *email*, *url*, etc. Aún así, tenga en cuenta los problemas de compatibilidad entre navegadores. Es posible que el comportamiento no sea el mismo en todos ellos, o que directamente alguno no funcione.
- Verificar el contenido de un control utilizando una expresión regular. Sin duda la opción más completa donde podemos especificar hasta el límite el formato de entrada de nuestros valores.

```
<script>
function datosOk(formulario) {
    var erNif = /^\\d{8}[A-Z]$/i; // mejorable ...
    var cajaTextoNif = formulario.nif;
    if(!erNif.test(cajaTextoNif.value)) {
        alert("NIF incorrecto");
        cajaTextoNif.focus();
        return false;
    }
    // otras validaciones ...
    return true;
}
</script>

...

<body>
    <form action='consultar.php' method='POST' name='f' onSubmit='return datosOk(this)'>
        NIF <input type='text' name='nif' id='idNif'>
        <input type='submit' value='Enviar'>
    </form>
</body>
```


Informar al usuario

Otro aspecto importante en la validación es cómo mostrar los errores al usuario. El código mostrado hasta el momento tiene una estructura que, aunque válida, realiza una validación secuencial de todos los controles, parando cuando encuentra un error, enviando el foco donde esté el problema y mostrándolo al usuario. De esta manera, cuando el usuario soluciona el problema, vuelve a realizarse el proceso, mostrando otro error, si existe.

Una mejora sería recorrer todos los controles y mostrar todos los errores posibles al mismo tiempo, enviando el foco al primer control que tiene un error. De esta manera, el usuario puede solucionar el problema en una única pasada.

Y para acabar comentaremos el carácter intrusivo de la instrucción *alert*. Este cuadro de texto es una ventana de tipo *modal* que para la ejecución del código y espera a que el usuario indique de alguna manera que continúe (pulsando Aceptar, p.e.).

Un enfoque más adecuado es informar al usuario de manera no intrusiva. Por ejemplo, mostrando mensajes de error en color rojo al lado de cada control. De esta manera, el usuario no tiene que aceptar una ventana, simplemente verá dónde hay un problema y lo solucionará.

Para hacer esto simplemente podemos preparar en los lugares adecuados elementos `` o `<P>` vacíos que asignaremos adecuadamente en el momento de la validación a través de la propiedad *innerHTML*.

Ejemplos sencillos

El siguiente código muestra en un cuadro de mensaje los enlaces de la propia página:

```
<html>
<head>
<title>Generación instantánea de índice de enlaces</title>
<script>
var textoEnlaces = "";
function mostrarEnlaces() {
    var item = 1;
    textoEnlaces = "Índice de enlaces:\n";
    for (n=0; n<document.links.length;n++) {
        textoEnlaces += item + ". " + document.links[n] + "\n";
        item++;
    }
    alert (textoEnlaces);
}
</script>
</head>
<body>
El <a href="http://www.cern.ch">World Wide Web</a> es esa conocida parte de
<a href="internet.html">internet</a> que se distribuye solamente por documentos hipertexto. <p>El
hipertexto es una forma de incorporar contenido multimedia en los documentos, con la característica
de crear enlaces, conocidos como <a href="hiperenlaces.html">hiperenlaces </a> a otros documentos.
<form>
    <input type='button' value='Índice' onClick='mostrarEnlaces()'/>
</form>
</body>
</html>
```

En este ejemplo podemos controlar que un link nos lleve a la página destino en función de alguna condición (por ejemplo, ser mayor de edad):

```
<html>
<head>
<title> Ejemplo de comprobación</title>
<script>
    function mayorDeEdad() {
        var strAñoDeNacimiento = prompt("Introduzca su año de nacimiento");
        var añoDeNacimiento = parseInt(strAñoDeNacimiento, 10);
        var fechaActual = new Date();
        var añoActual = fechaActual.getFullYear();
        var mayorDeEdad = ((añoActual-añoDeNacimiento)>18);
        if (!mayorDeEdad)
            alert("Hay que ser mayor para ir ahí dentro !");
        return mayorDeEdad;
    }
</script>
</head>
<body>
    <a href="http://www.microsoft.com" onclick="return mayorDeEdad();">Ir a Microsoft</a>
</body>
</html>
```

Si queremos intercambiar dos imagen al pasar el ratón por encima, haremos:

```
<img src='1.jpg' name='img' onmouseover="img.src='2.jpg'" onmouseout="img.src='1.jpg'">
```

En el siguiente ejemplo mostramos distintas imágenes (almacenadas en la carpeta imagenes) en función de la selección de una lista desplegable. El acceso se hace por el nombre del control.

```
<html>
<head>
<title> Galería de imágenes </title>
<script language="Javascript">
    function MostrarImagen(listaImagenes) {
        document.images.imagen.src=listaImagenes.options[listaImagenes.selectedIndex].value
    }
</script>
</head>
<body>
    <form name=f>
        Elige imagen: <br>
        <select name='listaImagenes' size='1' onChange='MostrarImagen(this)'\>
            <option value="imagenes\sargo.jpg">Sargo</option>
            <option value="imagenes\robaliza.jpg">Robaliza</option>
            <option value="imagenes\pulpo.jpg">Pulpo</option>
            <option value="imagenes\congrrio.jpg">Congrio</option>
        </select>
    </form>
    <img src='imagenes\sargo.jpg' name='imagen' width='700' height='500' border='0'>
</body>
</html>
```

En el siguiente ejemplo, se comprueba y se visualiza la selección de una lista múltiple:

```
<html>
<head>
<script>
function comprobar()
{
    var lista = document.getElementById("lista");
    var str="";
    var contador=0;
    for(i=0;i<lista.length;i++)
    {
        var opcion = lista.options[i];
        if (opcion.selected)
        {
            str += opcion.text + "(" + opcion.value + ")\n";
            contador ++;
        }
    }
    if (contador==0)
        alert("No se ha seleccionado nada");
    else
        alert (contador + " peces\n\n" + str);
}
</script>
</head>
<body>
<select multiple id="lista" size=6>
    <option value="R">Robaliza</option>
    <option value="S">Sargo</option>
    <option value="P">Pinto</option>
    <option value="M">Maragota</option>
    <option value="C">Congrio</option>
    <option value="A">Corobelo</option>
</select>
<input type=button value="info" onclick="comprobar()">
</body>
</html>
```

Librerías externas Javascript

jQuery

Software libre y de código abierto que proporciona unas funcionalidades basadas en el lenguaje JavaScript que facilitan enormemente el desarrollo de scripts, haciendo más con menos código. Básicamente, simplifica JavaScript.

Se basa en interactuar con la página mediante la función `$()` – alias de `jQuery()` – que recibe como parámetro un selector CSS o HTML, accediendo así a todos los nodos del árbol DOM seleccionados por la expresión, para aplicarles los métodos que sean necesarios.

Entre otras posibilidades, nos permite:

- Acceder y modificar el HTML a través del árbol DOM
- Acceder y modificar el CSS
- Programar los eventos de la página
- Realizar efectos y animaciones
- Utilizar AJAX
- JQuery UI nos proporciona multitud de plugins, controles (widgets), etc.



DataTables: plugin de tablas para jQuery

Permite generar tablas con búsqueda, filtros, orden y paginación.

Más en <https://datatables.net/>

Gráficos: Chart.js y CanvasJS



Permite incorporar a una página web multitud de gráficos de tarta, de barras, etc.

Más en <https://platzi.com/blog/plugins-jquery/>

AJAX

AJAX (Asynchronous JavaScript and XML) es una técnica que acelera la interacción con la web de manera drástica, puesto que permite actualizar elementos de una página sin tener que cargarla entera. Esto permite ahorrar mucho ancho de banda puesto que, utilizado convenientemente, se producen más llamadas al servidor pero este devuelve exactamente el contenido que se necesita actualizar.

Google hace desde siempre un uso importante de AJAX en sus webs: Gmail, Google Maps, Google Suggest (al autocompletar las búsquedas), etc.

Sin utilizar librerías externas (como JQuery, que simplifica enormemente el uso de AJAX), el código mínimo sería el siguiente (personalizándolo según el objetivo):

```
<html>
<head>
<script>
    function hacerAlgoConAJAX(valor) {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
                document.getElementById("idParrafo").innerHTML = xmlhttp.responseText;
            }
        }
        xmlhttp.open("GET", "getDesdeServidor.php?parametro=" + valor, true);
        xmlhttp.send();
    }
</script>
</head>
<body>

<input type="text" onkeyup="hacerAlgoConAJAX(this.value)">
<br />
<span id="idParrafo"></span>
</body>
</html>
```

Necesitaríamos además un script *getDesdeServidor.php* que generase la salida deseada (*echo*) para recuperar desde JavaScript y actualizar el elemento necesario (p, span, div, etc.).

En este caso se realiza la llamada asíncrona (es decir, se realiza la actualización de la página) cuando se pulsa una tecla en una caja de texto. Otro evento típico para desencadenar la actualización serían el *onChange* de un SELECT, para cambiar algo al seleccionar un elemento, por ejemplo, actualizar el contenido de otra lista sincronizada (marca/modelo, provincia/concello, etc.).

Anexo – MySQL desde phpMyAdmin

Introducción

phpMyAdmin es una herramienta escrita en PHP que nos ayuda, a través de un interfaz web, a trabajar con el gestor de bases de datos MySQL. Su licencia de uso es software libre, más en concreto GPL v2.

Su manejo es muy sencillo, y nos permite gestionar distintas bases de datos incluso en distintos servidores.

Creación de bases de datos.

Al crear una base de datos nos aparecen unos [conceptos un tanto extraños](#): cotejamiento (COLLATION), que se refiere al conjunto de reglas de comparación y de ordenación del texto, almacenado en un juego de caracteres concreto (CHARSET).

Tanto charset como collation se pueden asignar al propio gestor MySQL, a nivel base de datos, a una tabla o a un campo en concreto, heredándose en caso de no indicar lo contrario.

Tablas. Motores. Tipos de datos

Cuando estamos en la vista del servidor, tenemos una pestaña 'Motores' donde podemos ver todos los motores que podemos utilizar en el servidor de BD instalado.

Motor de almacenamiento	Descripción
FEDERATED	Federated MySQL storage engine
MRG_MYISAM	Collection of identical MyISAM tables
MyISAM	MyISAM storage engine
BLACKHOLE	/dev/null storage engine (anything you write to it disappears)
CSV	CSV storage engine
MEMORY	Hash based, stored in memory, useful for temporary tables
ARCHIVE	Archive storage engine
InnoDB	Supports transactions, row-level locking, and foreign keys
PERFORMANCE_SCHEMA	Performance Schema

Gestión de usuarios

Es fundamental la creación de distintos usuarios para distintas bases de datos. Si creamos todas las bases de datos con el usuario *root*, las aplicaciones tendrían que manejar este usuario que, en teoría, únicamente debería encargarse del mantenimiento básico del gestor.

Podemos gestionar usuarios desde la pestaña 'Usuarios' siempre que estemos situados sobre el servidor en la barra de navegación. En el caso de situarnos en una base de datos concreta, desde la pestaña 'Privilegios' accederemos a los usuarios con acceso a dicha base de datos.

Desde dicha pestaña, deberíamos cuanto antes seleccionar 'Editar los privilegios' para el usuario *root* de nuestro servidor de BD y cambiar su contraseña para asegurar la instalación. Posteriormente, para que el phpmyAdmin pueda entrar a consultarlo, debemos escribir dicha contraseña en el archivo *config.inc.php* situado en el raíz de la carpeta de instalación del phpmyAdmin.


Al añadir un usuario, en la categoría 'Base de datos para el usuario' tenemos varias opciones:

- No asignarle ninguna base de datos
- Crear base de datos con el mismo nombre y otorgar todos los privilegios. Con esto se crea una base de datos con el nombre del usuario y donde tiene todos los privilegios. Es una opción muy cómoda para muchas situaciones.
- Otorgar todos los privilegios al nombre que contiene comodín (username_) . Esta opción nos interesa en el caso de que queramos un usuario que pueda administrar varias bases de datos (cuyo nombre tiene que ser el nombre del usuario seguido por un guión bajo y luego lo que sea).
- Otorgar todos los privilegios para la base de datos '*nombreBD*'. Esta opción aparece si estamos creando un usuario directamente desde la pestaña 'Privilegios' de la BD.

Relaciones

Siempre que sea posible, y si el motor de almacenamiento elegido para las tablas involucradas lo permite (p.e. InnoDB), deberían crearse relaciones para mantener la integridad referencial.

Se pueden crear directamente desde SQL, o podemos utilizar el diseñador integrado en el phpmyAdmin. Accederemos a él desde la pestaña 'Diseñador', siempre que estemos situados en la barra de navegación en la base de datos correspondiente (ni en el servidor ni en una tabla).

Para crear una relación seleccionaremos el botón  que está etiquetado como 'Crear relación'. En ese momento debemos hacer clic primero en la clave primaria de la relación y a continuación en la clave foránea de la otra tabla.

En ese momento nos aparece una ventana 'Crear relación FOREIGN KEY' preguntándonos las acciones que debemos tomar en la clave foránea en el caso de eliminar (*on delete*) o modificar (*on update*) la clave primaria. Las posibles acciones (en ambos casos) son las siguientes: CASCADE, SET NULL, NO ACTION y RESTRICT (en MySQL NO_ACTION y RESTRICT son equivalentes).

Utilizaremos la opción adecuada en función de las características de las propias tablas y de su relación. Por ejemplo, eliminar en cascada será muy adecuado en la relación entre Pedidos y LíneasPedido, pues no parece tener mucho sentido mantener una línea de pedido sin su pedido relacionado. Establecer a NULL podría estar bien entre Clientes y Provincias, porque no por eliminar una provincia de la tabla voy a eliminar todos los datos de todos los clientes de esa provincia – mejor perder únicamente la relación con esa provincia estableciendo un *null* en dicho campo. Restringir una eliminación por ejemplo tendría sentido si quiero eliminar un cliente que tiene facturas. No debo eliminar la factura y tampoco el cliente, que es información muy importante para dicha factura.

Si no aparece esa ventana, y no se dibuja la línea de la relación, es que no ha sido creada por algún problema. Algunas de las situaciones más típicas son estas:

- No se ha usado InnoDB (o algún motor de almacenamiento que permita dichas relaciones)
- Si ya se han introducido registros en las tablas, verificar que éstos cumplen la integridad
- El campo de la clave foránea tiene que estar indexado
- Coinciden los tipos de datos de los dos campos a relacionar
- He establecido una relación con un SET NULL y el campo de la clave foránea no los permite

Exportación e Importación

La realización de exportaciones e importaciones de nuestras bases de datos son las operaciones básicas necesarias para realizar copias de seguridad. Aunque en muchos casos, éstas se podrán automatizar de múltiples formas (scripts del servidor, plugins de CMSs, etc.), es necesario saber hacerlo a mano desde esta herramienta.

Antes de realizar estas operaciones es importante fijarse en la barra de navegación superior, para ver dónde estamos situados. Dado que las pestañas 'Exportar' e 'Importar' siempre están visibles, dependerá de donde nos encontremos, para que las operaciones se refieran a distintos objetos:

- si estamos situados en el servidor, podremos importar o exportar todas (o algunas) de las bases de datos del servidor.
- si estamos situados en una base de datos, podremos importar o exportar todas (o algunas) de las tablas de dicha base de datos
- si estamos situados en una tabla, podremos importar o exportar todos (o algunos) de los registros de dicha tabla.

La exportación se puede realizar en diversos formatos, aunque el más completo (sobre todo para futuras importaciones) es SQL. Entre las distintas opciones está SQL, CSV, JSON, ArrayPHP, XML y diversos formatos ofimáticos como por ejemplo hojas de cálculo, Latex, ODT, etc.

Independientemente de esto, siempre que realicemos una exportación tendremos una primera opción 'Rápida' que generará un archivo con el SQL necesario para importar posteriormente todo el contenido y restricciones existentes.

Existe una segunda opción 'Personalizado' en donde podremos afinar la exportación variando por ejemplo:

- nombre de los archivos generados
- juego de caracteres utilizado
- compresión utilizada
- crear estructura y/o volcar datos
- inclusión de comentarios
- distintas posibilidades en la sintaxis del SQL generado
- longitud máxima de las consultas
- conversión de formatos de fechas (TIMESTAMP > UTC)