DESARROLLO WEB EN ENTORNO SERVIDOR

GRADO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

IG Formación
www.igformacion.com

Este manual es una pequeña introducción al lenguaje de programación PHP. Pretende ser una guía de apoyo para los estudiantes del módulo profesional Desarrollo Web en Entorno Servidor, del Grado Superior de Desarrollo de Aplicaciones Web (DAW).

El estudiante que utilice este manual tiene la posibilidad de adquirir los conocimientos básicos de la programación con PHP. La extensión de los contenidos incluidos hace imposible el desarrollo completo en la mayoría de los casos.

Los contenidos incluidos en este manual abarcan los conceptos básicos y las técnicas habituales para el desarrollo de aplicaciones web. Además, se acompaña de ejemplos que sirven para ilustrar dichos conceptos y técnicas.

Se abordan los puntos principales relacionados con el uso del lenguaje PHP que intercalan su código con el de las páginas web, ofreciendo una descripción detallada de su sintaxis y de las estructuras y funciones primordiales. Se presta una atención especial a los aspectos de conexión y acceso a bases de datos desde las aplicaciones web del servidor. También se hace una introducción al patrón de diseño Modelo-Vista-Controlador (MVC).

Todos los capítulos incluyen ejemplos con el propósito de facilitar la asimilación de los conocimientos tratados.

Este manual se distribuye sin garantía de ningún tipo, ya sea implícita o explícita. Aun así, la intención es que resulte útil, así que rogamos que cualquier error que encuentre nos sea comunicado para su subsanación.

NOTA SOBRE LOS EJERCICIOS

Es conveniente que leáis el punto "Formularios HTML y PHP" antes de empezar a hacer los ejercicios, pues será necesario que conozcáis cómo pedir datos al usuario.

Contenido

Comenzando con PHP

PHP es un lenguaje de programación basado en *secuencia de comandos* o *sentencias*. Estas secuencias de comandos componen nuestro *programa* o *script*, que preprocesará el *analizador sintáctico* (*parser)* de PHP y generará el correspondiente código, generalmente una página HTML que enviará a nuestro navegador.

Esto nos permite la generación de páginas Web dinámicas. Un ejemplo bastante simple es la fecha del copyright de nuestro sitio web. Mediante un pequeño programa en PHP podemos visualizar correctamente el año en curso.

```
Copyright &copy; <?php echo date('Y'); ?>
```

En esta lección comenzaremos con la programación en PHP. Analizaremos el formato del código. Aprenderemos las reglas generales de la sintaxis del PHP y como crear comentarios.

Finalmente, aprenderemos buenas prácticas para hacerte la vida más fácil y mejor tu código.

La sintaxis en PHP

El primer paso para utilizar PHP es poner tu propio código PHP en un archivo con la extensión `.php`. Esto hará que el servidor esté atento y le indique que tiene que procesar código en PHP.

Un bloqueo de PHP código tiene la siguiente sintaxis.

Sintaxis de código escrito en PHP

```
<?php

Código escrito en PHP.

?>
```

Si el archivo sólo contiene PHP, o si termina con código PHP, podemos dejar sin poner la etiqueta de cierre `?>`. Si hay alguna información después de `?>`, como líneas extra o espacios en blanco, será interpretada como HTML y enviada al navegador. Esto dará espacios en blanco al final de nuestra página web y posibles errores.

En algunos servidores se permite la etiqueta corta de inicio `<?` para empezar el código PHP, pero no es conveniente hacerlo. Si trasladamos nuestros programas a otro servidor que no lo admita, nos exponemos a que no funcione correctamente.

Añadiendo PHP a nuestra página web

Podemos tener un bloque PHP que abarque un archivo entero o un bloque PHP que se intercale con HTML. Si intercalamos no se nos debe olvidar cerrar el código PHP con la etiqueta `?>`, y siempre terminar cada sentencia con un punto y coma.

```
<html>
<head>
    <title>Añadiendo PHP a nuestra página web</title>
</head>
<body>
    <h1>Hola mundo</h1>
    <p>Copyright &copy; <?php echo date('Y'); ?></p>
</body>
```

```
</html>
```

En el ejemplo anterior hemos intercalado PHP en una página web HTML. Se puede observar que para mostrar la información utilizamos la función `echo`. Esta función mostrará el texto que le estamos pasando como argumento.

```
<?php
$html =
'<html>
<head>
<title>Añadiendo PHP a nuestra página web</title>
</head>
<body>
<h1>Hola mundo</h1>
<p>Copyright &copy; ' . date('Y') . '</p>
</body>
</html>';

echo $html;
?>
```

Esta segunda forma de codificar con PHP consiste en generar la cadena que contiene todas las etiquetas HTML para después mostrarlas. Esta forma es bastante más engorrosa, y bajo mi punto de vista, totalmente desaconsejable.

Los comentarios

Los comentarios son una parte importante del código en PHP. Son anotaciones de qué y cómo estamos haciendo nuestro script. Aunque recordemos mañana que estabas intentando hacer hoy viendo una pequeña cantidad de código, probablemente dentro de seis meses no seamos capaces. Siempre se intentará documentar lo que se está tratando de hacer mientras lo hacemos.

Los comentarios no se envían al navegador y no ralentizan el sistema, pues no son analizados por PHP. Si miramos el código fuente de una página web, solo veremos los comentarios HTML, no los de PHP.

Existen, principalmente, dos tipos de comentarios en PHP. El primer tipo es para comentar un línea simple o parcial. Empieza con // y termina al final de la línea.

```
<?php
$ciudad      = 'Alicante'; // Esto es un comentario de una línea.
$temperatura = 'alta';

echo '<p>La temperatura en ' . $ciudad . ' es muy ' . $temperatura . '.</p>';

// Podemos tener comentario en varias líneas
// simplemente poniendo comentarios parciales.
?>
```

El segundo comentario es el de bloque. El comentario de bloque comienza con `/*` y termina con `*/`.

```
<?php
$ciudad   = 'Alicante'; /* Esto es un comentario de una línea. */
$tipoTemp = 'alta';

echo '<p>La temperatura en ' . $ciudad . ' es muy ' . $tipoTemp . '.</p>';

/*
Podemos tener comentario en varias líneas porque, hasta que no pongamos
el final de comentario se ignorará todo.
*/
?>
```

Observa que, aunque el comentario sea de una sola línea, debemos indicar cuando finaliza el mismo.

Existe un tipo especial de comentario multilínea llamado PHPDoc. Es muy útil para definir y comentar nuestro código y a partir de generadores de documentos, como phpDocumentor[1], poder hacer la documentación de nuestro API.

El comentario PHPDoc comienza con `/**`, tiene un `*` al comienzo de cada línea y finaliza con `*/`. Este es un ejemplo que documenta el mismo archivo.

```
/**
* Descripción breve del archivo
*
* Descripción larga del archivo...
*
* @version 1.2 2011-02-03
* @package Nombre del proyecto
* @copyright Copyright (c) 2017 Nombre de la empresa
* @license GNU General Public License
* @since Since Release 1.0
*/
```

Formateando el código

Los estilos hacen que el código sea más fácil de leer y comprender, porque organiza la información y nos indica qué esperamos. Si no hay párrafos, si todas las sentencias están puestas una tras otra sin cortes, podemos leerlas, pero es más difícil.

Se suele formatear a la vez que se programa. El código se ejecuta bien sin necesidad de formatearlo, pero es más difícil para nosotros leerlo. Es más difícil ver qué está sucediendo y encontrar errores.

Un ordenador no tiene problema en leer el siguiente código.

```
<?php $mensaje='';$nombre=filter_input(INPUT_POST,'task',FILTER_SANITIZE_STRING);if
($nombre!='')$mensaje="Buenos días $nombre.";else $mensaje='Buenos días Sr.';echo
mensaje?>
```

Sin embargo, aunque tú no conozcas PHP si el código está formateado, seguirlo es fácil, otra cosa es entenderlo.

```
<?php
$mensaje = '';
$nombre  = filter_input(INPUT_POST, 'nombre', FILTER_SANITIZE_STRING);

if ($nombre != '')
    $mensaje = "Buenos días $nombre.";
else
    $mensaje = 'Buenos días Sr.';

echo $mensaje;
?>
```

Al formatear nuestro código debemos tener en cuenta las siguientes cuestiones:

- Sangrado: ¿Qué parte de nuestro código debería ser sangrado? ¿Cuándo utilizar el sangrado? ¿Cuántos espacios debería tener la línea que hemos sangrado? ¿Utilizamos espacios en blanco o tabulador?

[1] Para más información visita la web: www.phpdoc.org

- Longitud de la línea: ¿Cuántos caracteres debería tener una línea de código? ¿Restringiremos la longitud de línea a una sola cuando puede ser más de una?

- Espacios en blanco: ¿Qué cantidad de espacios en blanco adicionales (esto es, espacios, nuevos o vacíos) añadiremos para conseguir fluidez o legibilidad?

- Uso de las llaves de conjunto (`{}`): Si estos son opcionales en la sintaxis, ¿los utilizaremos de todas formas?

Todas estas cuestiones las iremos contestando a lo largo que desarrollaremos nuestro código.

Existen varios estilos populares para el formateo del código PHP:

- Zend Framework[2].

- Pear Coding Standards[3].

- PHP-Fig Commnuity[4].

Cada código suele tener un estilo diferente a los demás. Hay diferentes estilos de formateo de código PHP, pero lo más importante es que seamos coherentes con nuestro propio estilo. A lo largo de todos los ejemplos iremos haciéndonos con nuestro propio estilo.

Buenas prácticas

Usando buenas prácticas harán que nuestro código sea menos propenso a errores, más fácil de mantener y más seguro.

Ya hemos aprendido algo de buenas prácticas en este capítulo:

- No utilizaremos las etiquetas cortas de inicio de PHP `<? ... ?>`, en su lugar utilizamos `<?php ... ?>`

- Comentaremos el código, pero no todo, solo aquello que consideremos necesario para hacer más compresible o porqué hemos desarrollado el mismo de esa forma.

- Seremos congruentes en nuestro propio estilo de código, pero reconozcamos que otros programadores pueden tener sus propios estilos.

- Si trabajamos en un proyecto con otros programadores, utilizaremos el estilo adoptado por el proyecto en lugar de nuestro estilo personal.

Además, deberíamos:

- Poner la etiqueta final, corchete o paréntesis cuando creemos la etiqueta de inicio, corchete, o paréntesis. Nos evitaremos muchos problemas.

- En los bucles no olvidarse poner la condición de salida de este.

- Utilizaremos líneas extra para separar bloques de código.

[2] https://framework.zend.com/manual/1.12/en/coding-standard.html
[3] https://pear.php.net/manual/en/standards.php
[4] http://www.php-fig.org/psr/psr-1/

- Sangraremos elementos anidados.
- Seremos congruentes con la nomenclatura convencional y utilizaremos nombres significativos.
- Activaremos los informes de error mientras estemos desarrollando. Esto permite localizar y mostrarnos los errores.
- Desactivaremos los informes de erro cuando nuestro programa esté en producción. Evitará mostrar mensajes de error incomprensibles para los usuarios.

Todo lo visto hasta ahora lo pondremos en práctica a medida que vayamos avanzando.

Ejercicios

1. Realiza un trabajo sobre la codificación en Zend Framework indicando:
   - Cómo se formatea un fichero en PHP
   - La convención sobre los nombres.
   - Estilo de codificación.

Palabras reservadas

Las palabras reservadas[5] en PHP se utilizan para construir el lenguaje y definen métodos, funciones, clases, etc. Al ser reservadas no podemos utilizarlas como nombres de variables o funciones. A continuación, podemos ver algunas palabras reservadas de PHP.

Palabra	Tipo	Descripción
`and`		Operador lógico Y.
`array()`	Función	Permite crear un vector.
`as`		Asigna variable de referencia en un ciclo `foreach`.
`break`		Finaliza la ejecución de una sentencia `for`, `foreach`, `while`, do-while o `switch`.
`case`		Coincidencia de valor en una sentencia `switch`.
`catch`		En la sentencia try parte del código que se ejecuta cuando se produce una excepción.
`class`		Define una clase.
`clone`		Crea una copia de un objeto.
`const`		Define una constante.
`continue`		Salta el resto de la iteración actual de un bucle y continúa con la siguiente iteración.
`die()`	Función	Finaliza la ejecución de un programa. Es equivalente a `exit()`.
`do`		Se utiliza en la sentencia `do/while`.
`echo`		Imprime en pantalla una cadena de caracteres.
`else`		Condición falsa en una sentencia `if/else`.
`elseif`		Condición falsa en una sentencia `if/elseif`, que extiende a una nueva condición.
`empty()`	Función	Determina si una variable se encuentra vacía.

`endfor;`
`endforeach;`
`endif;`
`endwhile;`
`endswitch;`
		Alternativa para estructuras de control.

`exit()`	Función	Finaliza la ejecución de un programa.
`final`		Impide que clases hijas sobrescriban un método.
`finally`		En la sentencia try parte del código que siempre se ejecuta.
`for`		Realiza un número determinado de iteraciones.
`foreach`		Permite recorrer vectores.
`function`		Declara una función.
`global`		Permite acceder a variables declaradas fuera de una función.
`if`		Condición verdadera en una sentencia `if/else`.
`include`	Función	Incluye y evalúa un archivo en nuestro código.
`include_once`	Función	Similar a `include`, pero si el archivo ya ha sido incluido no volverá a incluirlo.

[5] http://php.net/manual/es/reserved.keywords.php

`instanceof`		Determina si una variable es un objeto instanciado de una clase.
`interface`		Especifica que métodos deben ser implementados por una clase.
`isset()`	Función	Determina si una variable está definida y no es `null`.
`list()`	Función	Asigna variables como si fuesen un vector.
`namespace`		Proporciona una forma de agrupar funciones, clases, interfaces que se encuentran relacionadas.
`new`		Instancia una clase para crear un objeto.
`or`		Operador lógico O.
`print`		Muestra una cadena de caracteres.
`private`		Las propiedades o métodos de una clase solo son accesibles desde la clase que los ha definido.
`public`		Las propiedades y métodos de una clase son accesibles desde donde sea.
`protected`		Las propiedades y métodos de una clase son accesibles solo desde la misma clase o clases heredadas.
`require`	Función	Similar a `include`, solo que en caso de fallar produce un error que detiene el programa.
`require_once`	Función	El archivo no es incluido si ya se ha realizado con anterioridad.
`return`		Devuelve el control del programa al módulo que lo invoca.
`static`		Establece la existencia de una variable en el ámbito local de una función impidiendo que se destruya cuando finaliza la ejecución de la función.
`switch`		Permite evaluar múltiples condiciones.
`throw`		Lanza una excepción.
`try`		Permite la captura de excepciones, esta instrucción está acompañada por `catch` o `finally`.
`unset()`	Función	Destruye una variable.
`var`		Utilizado para declarar propiedades. Esta instrucción está *obsoleta* o *deprecated*.
`while`		Realiza un bucle un número determinado de veces o mientras se cumpla una condición.
`xor`		Operador lógico XOR.

Tabla 1. Palabras reservadas en PHP

Variables

En este capítulo aprenderemos que son las variables, cómo se definen y cómo utilizarlas en cada caso. Aprenderemos a trabajar con variables denominadas simples, como texto y números.

Introducción a las variables

Las *variables* se utilizan para almacenar información que puede cambiar a lo largo de nuestro programa. Mediante la creación de variables, disponemos de un método para escribir un programa que puede ser utilizado con diferentes datos.

En PHP, las variables empiezan con un signo de dólar (`$`) y se le asigna un valor utilizando el *operador de asignación igual* (=).

Veamos un ejemplo.

```
<?php
$nombre    = 'Javier';
$apellidos = 'Miras';
?>
```

La siguiente lista incluye reglas para nombrar variables, después del signo del dólar:

- Debemos empezar las variables con una letra o un guión bajo. Por convenio, el guión bajo se usa sólo en ciertas ocasiones. Por ahora, empezaremos siempre con una letra.
- Sólo podemos utilizar caracteres y guión bajo (`a-z`, `A-Z` y _).
- No podemos utilizar espacios ni símbolos especiales, como `#`, `*` o `/`.
- Si la variable tiene más de una palabra, debemos separarlas con letra mayúscula o guión bajo; por ejemplo, `$nombreApellidos` o `$nombre_apellidos`.
- La longitud de las variables debe ser la apropiada. Lo suficientemente larga para que sea descriptiva, a la vez que manejable para trabajar con ella.
- El nombre de la variable debe tener sentido y aportar información de lo que va a contener. Si necesitamos poner un comentario a la variable, el nombre es incorrecto.
- Evitaremos nombres que creen confusión. Si el nombre nos lleva a una ambigüedad en cuanto a lo que contiene una variable, el nombre es incorrecto.

Las variables son sensibles a mayúsculas y minúsculas, por lo que, `$nombreapellidos`, `$nombreApellidos`, `$Nombreapellidos` y `$NombreApellidos`, son cuatro variables distintas.

En algunos lenguajes de programación tenemos que declarar la variable previamente y decir qué tipo de información va a contener, texto o números. En PHP no necesitamos hacer esto para las variables simples. La variable lleva el tipo de información que le asignemos.

Es importante inicializar la variable antes de utilizarla o tendremos un error de variable indefinida.

Las variables complejas, cada matrices y objetos, debemos declararlas antes.

Nulo

El valor *nulo* (*null*[6]) representa a una variable a la que:

- todavía no se le ha dado valor,
- se ha destruido mediante la función `unset()`[7] o
- se le ha asignado el tipo `null`.

Utilizaremos `null` o `NULL`, puesto que es insensible a mayúsculas y minúsculas, pero siempre hay que seguir un criterio de estilo.

Para verificar que una variable es nula utilizaremos la función `is_null()`[8]. Esta función nos devolverá verdadero si es nula o falso si no lo es.

```
<?php
$casado = null;

if (is_null($casado))
    echo '$casado es nulo';
?>
```

Booleanos

El *booleano* (*boolean*[9]) es el tipo más simple. Puede tomar dos valores: *Verdadero* (`true`) o *falso* (`false`). Los valores `true` o `false` son insensible a mayúsculas o minúsculas; manteniendo siempre un criterio de estilo.

```
<?php
$casado     = true;
$divorciado = false;
$trabaja    = True;
$cobraMucho = FALSE;
?>
```

Cuando trabajamos con booleanos hay que tener en cuenta que se considerará falso las siguientes conversiones:

- El cero, ya sea entero (0), real (0.0) o cadena ("0").
- La cadena vacía ("").
- Un vector de vacío.
- Un objeto sin variables miembro.
- El valor nulo.

Hay que tener cuidado con el valor falso cuando lo queramos visualizar con `echo`, pues nos devolverá una cadena vacía.

[6] http://php.net/manual/es/language.types.null.php
[7] http://php.net/manual/es/function.unset.php.
[8] http://php.net/manual/es/function.is-null.php.
[9] http://php.net/manual/es/language.types.boolean.php

Las operaciones lógicas que podemos realizar se muestran a continuación.

Operador	Descripción	Ejemplo	Resultado
`and ó &&`	Y	`$a and $b`	Verdadero si $a y $b son verdadero, false en el resto de los casos.
`or ó ||`	O inclusiva	`$a or $b`	Falso si $a y $b son falso, verdadero en el resto de los casos.
`xor`	O exclusiva	`$a xor $b`	Verdadero si $a o $b son verdadero, pero no ambos a la vez.
`not ó !`	Negación	`$a = !$a`	Verdadero si $a es falso, y viceversa.

Tabla 2. Operadores lógicos.

Los operadores lógicos nos permitirán crear condiciones que veremos más adelante en las estructuras condicionales y repeticiones.

Otro tipo de operaciones que podemos hacer con los booleanos son las de comparación.

Operador	Descripción	Ejemplo	Resultado
==	Igual	`$a == $b`	Verdadero si $a es igual a $b, falso en caso contrario.
>	Mayor que	`$a > $b`	Verdadero si $a es mayor que $b, falso en caso contrario.
>=	Mayor o igual que	`$a >= $b`	Verdadero si $a es mayor o igual que $b, falso en caso contrario.
<	Menor que	`$a < $b`	Verdadero si $a es menor que $b, falso en caso contrario.
<=	Menor o igual que	`$a <= $b`	Verdadero si $a es menor o igual que $b, falso en caso contrario.
<> ó !=	Distinto	`$a <> $b` ó `$a != $b`	Verdadero si $a es distinto a $b, falso en el resto de los casos.
===	Estrictamente igual	`$a === $b`	Verdadero si $a es igual a $b y son del mismo tipo, falso en caso contrario.
!==	Estrictamente distinto	`$a !== $b`	Verdadero si $a es distinto a $b o son de tipos distintos, falso en el resto de los casos.
<=>	Combinación de operador de comparación (Nave espacial)	`$a <=> $b`	Si $a > $b, devuelve 1 Si $a < $b, devuelve -1 Si $a = $b, devuelve 0

Tabla 3. Operadores de comparación.

A lo largo del libro iremos viendo cada uno de estos operadores, tanto los lógicos como los de comparación.

Cadenas de texto

En programación, otro término para texto es *cadena de caracteres* (*string*[10]). "Hola a todos" es una cadena de caracteres. Para asignárselo a una variable se utiliza el *operador de asignación*, que es el signo de igual (=).

```
<?php
$cadena = 'Hola a todos';

echo $cadena;
?>
```

Las cadenas de caracteres empiezan y terminan con comillas simples (`'`) o comillas dobles (`"`). Pero existe una gran diferencia entre utilizar una u otra.

```
<?php
$nombre = 'Javier';
$cadena = 'Hola $nombre';

echo $cadena;
?>
```

A la variable `$nombre` se le asigna la cadena de caracteres 'Javier'. Después a la variable `$cadena` se le asigna la cadena de caracteres 'Hola $nombre'. La visualización dará como resultado 'Hola $nombre'.

```
<?php
$nombre = 'Javier';
$cadena = "Hola $nombre";

echo $cadena;
?>
```

De forma análoga, pero en este caso a la variable `$cadena` se le asigna "Hola $nombre", con comillas dobles. La visualización dará como resultado 'Hola Javier'.

Al utilizar comillas dobles, PHP interpretará y analizará la cadena, por si tiene que expandir la variable. Mientras que con comillas simples se mostrará literalmente la cadena.

Hay que tener cuidado cuando se utilizan comillas dobles y se nos den casos como el mostrado a continuación.

```
<?php
$vehiculo1 = 'coche';
$vehiculo2 = 'moto';

$cadena1 = "Tengo 2 $vehiculo1s";   // Error $vehiculo1s no está definida.
$cadena2 = "Tengo 3 {$vehiculo2}s"; // Bien. Se visualizará como motos.

echo $cadena2;
?>
```

Siempre que inmediatamente después de una variable haya que poner una letra, debemos poner la variable entre llaves (`{}`).

Si quisieras poner un texto entrecomillado, hay que utilizar la *barra invertida* (*backslah*). Con ello le indicamos a PHP que lo que viene a continuación no es un *carácter de control* sino un literal y no lo analizará.

```
<?php
$vehiculo1 = 'coche';
```

[10] http://php.net/manual/es/language.types.string.php

```
$vehiculo2 = 'moto';

$cadena1 = "Tengo 2 "$vehiculo1s"";      // Incorrecto.
$cadena2 = "Tengo 3 \"{$vehiculo2}s\"";  // Correcto.

echo $cadena2;
?>
```

En la asignación de la `$cadena1` se producirá un error de sintaxis, pues ha encontrado algo inesperado como `$vehiculo1s`. Por el contrario, en la asignación de la `$cadena2` no se produce ningún error al indicar que trate las dobles comillas como un literal (`\"`).

Una solución que te evitará numerosos problemas es la combinación de comillas simples con comillas dobles y viceversa.

```
<?php
$cadena1 = 'Tengo 2 "coches"';
$cadena2 = "Tengo 3 'motos'";
?>
```

Concatenado cadenas

La *concatenación* (*concatenation*) es la unión de dos cadenas de caracteres. Para ello utilizaremos el punto (.).

```
<?php
$nombre = 'Javier';
$cadena = 'Hola ' . $nombre;

echo $cadena;
?>
```

Observar el espacio que se ha dejado detrás de "Hola ", esto nos permitirá visualizar bien el resultado de la concatenación "Hola Javier". Es importante para que no quede junto a la variable `$nombre` y de como resultado "HolaJavier".

```
<?php
$nombre   = 'Javier';
$apellido = 'Miras;

$cadena1  = 'Hola ' . $nombre . ' ' . $apellido . '.';
$cadena2  = "Hola $nombre $apellido.";
?>
```

A la variable `$cadena1` le hemos concatenado un punto al final. Para ello, el punto lo hemos puesto como si fuese una cadena de caracteres.

A la variable `$cadena2`, a pesar de que lleve inmediatamente después un carácter (.), no hay que poner la variable entre llaves. Un punto no puede formar parte del nombre de una variable y PHP no lo interpretará como tal, sino como un literal.

En ambos casos el resultado será "Hola Javier Miras."

```
<?php
$nombre = 'Javier';
$nombre .= ' Miras'; // Equivalente a $nombre = $nombre . ' Miras';
?>
```

El operador "`.=`" es una contracción de concatenar y después asignar.

PHP proporciona numerosas funciones[11] para la manipulación de cadenas. Algunas de ellas las utilizaremos más adelante.

Números

En PHP podemos trabajar con dos tipos numéricos: Enteros[12] y reales[13]. En ambos casos tendremos números positivos y negativos.

Mediante la constante PHP_INT_SIZE podemos determinar el tamaño del entero. También podemos saber el valor máximo con PHP_INT_MAX y el valor mínimo con PHP_INT_MIN.

Para asignarle valor a una variable numéricas se utiliza el operador de *operador de asignación*.

```
<?php
$a = 3;
$b = PHP_INT_MIN;   // -9223372036854775808
$c = PHP_INT_MAX;   //  9223372036854775807

$d = $a + $b + $c;
?>
```

El resultado será 2.

```
<?php
$c = 2.25;

echo $c;
?>
```

La función `echo()`[14] nos permite visualizar el resultado 2.25.

Las operaciones aritméticas que podemos realizar se muestran a continuación.

Operador	Descripción	Ejemplo	Resultado
+	Suma	`$a = 3 + 5`	8
-	Resta	`$a = 3 - 5`	-2
*	Multiplicación	`$a = 3 * 5`	15
/	División	`$a = 3 / 5`	0.6
%	Resto	`$a = 3 % 5`	3
++	Incremento	`$a++`	4
--	Decremento	`$a--`	3
**	Exponenciación	`$a = 3**5`	243

Tabla 4. Operaciones aritméticas.

Dentro de los operadores de incremento y decremento nos encontramos con el post y preincremento, y el post y predecremento.

Operador	Descripción	Ejemplo	Resultado
++	Postincremento	`$a++`	Primero devuelve el valor y luego lo incrementa.

[11] http://php.net/manual/es/book.strings.php
[12] http://php.net/manual/es/language.types.integer.php.
[13] http://php.net/manual/es/language.types.float.php.
[14] http://php.net/manual/es/function.echo.php

	Preincremento	`++$a`	Primero incrementa el valor y luego devuelve el valor.
`--`	Postdecremento	`$a--`	Primero devuelve el valor y luego lo decrementa.
	Predecremento	`--$a`	Primero decrementa el valor y luego devuelve el valor.

Veamos un ejemplo.

```
<?php
$a = 5;

echo '$a++ vale ' . $a++ . ', pues primero devuelve el valor y luego se incrementa.
      El valor de $a ahora es ' . $a;

echo '<br/>';

echo '++$a vale ' . ++$a . ', pues primero se incrementa el valor y luego se
     devuelve. El valor de $a ahora es ' . $a;
?>
```

Con los números también podemos realizar las siguientes operaciones de asignación.

Operador	Ejemplo	Equivale
+=	`$a += 5`	`$a = $a + 5`
-=	`$a -= 5`	`$a = $a - 5`
***=**	`$a *= 5`	`$a = $a * 5`
/=	`$a /= 5`	`$a = $a / 5`
%=	`$a %= 5`	`$a = $a % 5`

Tabla 5. Operaciones de asignación.

Hay que tener cuidado cuando trabajamos con números. Si estamos trabajando con número enteros y sobrepasamos los límites, el resultado obtenido es un número real.

```
<?php
$a = PHP_INT_MAX;
$b = PHP_INT_MAX;

$c = $a + $b;
?>
```

El resultado será 1.844674407371E+19.

De igual forma, si operamos con número enteros y reales el resultado será un número real.

PHP proporciona una gran cantidad de funciones matemáticas[15].

Cadenas y números juntos

Aunque operar con cadenas y números no es lo más lógico, si se hace, hay que tener cuidado.

```
<?php
$fechaNacimiento = '08/02/....';
$numero          = 2;

$c = $fechaNacimiento * $numero;
?>
```

[15] http://php.net/manual/es/book.math.php.

En este primer caso el resultado será 16. Al hacer la multiplicación PHP convierte la cadena de caracteres `$fechaNacimiento` en el número 8.

```
<?php
$fechaNacimiento = 'ocho de febrero de ....';
$numero          = 2;

$c $fechaNacimiento * $numero;
?>
```

El resultado en este segundo caso será cero. PHP intentará convertir la cadena de caracteres `$fechaNacimiento`, al no poder obtener un número coherente devuelve cero.

```
<?php
$fechaNacimiento = 802;
$numero           = 2;

$c = $fechaNacimiento . $numero;
?>
```

En este último caso el resultado será 8022. PHP convierte en cadena de caracteres las dos variables numéricas, `$fechaNacimiento` y `$anyo`, para poder hacer la concatenación.

Orden de precedencia de los operadores

Cuando trabajamos con operadores debemos tener en cuenta su orden de precedencia y su asociatividad. El primero indica el orden de la operación en sentido vertical y el segundo en sentido horizontal.

Operador	Asociatividad	Precedencia
()	No	Mayor
++, --	Derecha	
not ó !	Derecha	
*, / ó %	Izquierda	
+, - ó .	Izquierda	
<, <=, >, >=, <>	No	
==, !=, ===, !==	No	
&&	Izquierda	
||	Izquierda	
=, +=, -=, *=, .=, !=, &=	Derecha	
and	Izquierda	
xor	Izquierda	
or	Izquierda	Menor

Tabla 6. Orden de precedencia de los aperadores

Si no queremos complicarnos la vida, utilizaremos paréntesis.

Vectores y matrices

Un *vector* (*array*[16]) contiene múltiples valores en una variable simple. Dentro de una variable vector tenemos una lista entera de valores. Podemos acceder al vector completo mediante su nombre, como lo haríamos con un variable simple, o a cada uno de sus valores mediante un índice.

Podemos anidar vectores dentro de otros vectores, que llamaremos *vectores multidimensionales* o *matrices.*

```
<?php
$provincia1 = 'Alicante';
$provincia2 = 'Castellón';
$provincia3 = 'Valencia';
?>
<html>
<head>
    <title>Comunidad Valenciana</title>
</head>

<body>

    <h1>Comunidad Valenciana</h1>
    <p><?php echo $provincia1; ?></p>
    <p><?php echo $provincia2; ?></p>
    <p><?php echo $provincia3; ?></p>

</body>

</html>
```

Para mostrar las tres provincias que componen la Comunidad Valenciana hemos tenido que emplear tres variables simples.

```
<?php
$provincias = array('Alicante', 'Castellón', 'Valencia');
?>
<html>
<head>
    <title>Comunidad Valenciana</title>
</head>

<body>

    <h1>Comunidad Valenciana</h1>
    <p><?php echo $provincias[0]; ?></p>
    <p><?php echo $provincias[1]; ?></p>
    <p><?php echo $provincias[2]; ?></p>

</body>

</html>
```

Ahora hemos utilizado un vector, al que hemos llamado `$provincias`, para almacenar las tres provincias de la Comunidad Valenciana. Para acceder a cada una de ellas utilizaremos los corchetes y el índice al que queremos acceder. Hay que tener en cuenta que el índice del primer elemento de un vector es el 0.

Existe otro tipo de vector, es el llamado *vector asociativo*. En un vector asociativo en lugar de acceder mediante un índice numérico accederemos a través de una *clave* (*key*).

```
<?php
$capitales = array('Alicante' => 'Alicante',
                   'Castellón' => 'Castellón de la plana',
                   'Valencia' => 'Valencia');
?>
<html>
```

[16] http://php.net/manual/es/language.types.array.php

```
<head>
    <title>Capitales de la Comunidad Valenciana</title>
</head>

<body>

    <h1>Comunidad Valenciana</h1>
    <p><?php echo $capitales['Alicante']; ?></p>
    <p><?php echo $capitales['Castellón']; ?></p>
    <p><?php echo $capitales['Valencia']; ?></p>

</body>

</html>
```

Observa la forma de crear un vector asociativo. A cada una de las claves se le asocia su valor mediante el *símbolo de implicación* (=>).

Para ver el contenido de un vector nos será más útil la función `print_r()`[17] que hacer un `echo()`.

```
<?php
$capitales = array('Alicante' => 'Alicante',
                   'Castellón' => 'Castellón de la plana',
                   'Valencia' => 'Valencia');

print_r($capitales);
?>
```

Cuando necesitemos un vector de más de una dimensión tendremos que utilizar una *matriz* (*matrix*[18]).

```
<?php
$comunidadValenciana =
    array(array('provincia' => 'Alicante', 'capital' => 'Alicante'),
          array('provincia' => 'Castellón', 'capital' => 'Castellón de la plana'),
          array('provincia' => 'Valencia', 'capital' => 'Valencia'));

print_r($comunidadValenciana);
?>
```

Para acceder a un elemento en particular de la matriz de dos dimensiones utilizaremos dos índices, el primero para acceder a un elemento del vector y el segundo para acceder a los elementos que hay dentro de ese vector. En el caso de que fuese de tres dimensiones, serían tres índices y así sucesivamente.

```
<?php
$comunidadValenciana =
    array(array('provincia' => 'Alicante', 'capital' => 'Alicante'),
          array('provincia' => 'Castellón', 'capital' => 'Castellón de la plana'),
          array('provincia' => 'Valencia', 'capital' => 'Valencia'));
?>
<html>
<head>
    <title>Comunidad Valenciana</title>
</head>

<body>

    <h1>Comunidad Valenciana</h1>

    <h2>Provincias</h2>
    <p><?php echo $comunidadValenciana[0]['provincia']; ?></p>
    <p><?php echo $comunidadValenciana[1]['provincia']; ?></p>
    <p><?php echo $comunidadValenciana[2]['provincia']; ?></p>

    <h2>Capitales</h2>
    <p><?php echo $comunidadValenciana[0]['capital']; ?></p>
```

[17] http://php.net/manual/es/function.print-r.php
[18] Es común utilizar también la palabra vector o array para las matrices.

```
        <p><?php echo $comunidadValenciana[1]['capital']; ?></p>
        <p><?php echo $comunidadValenciana[2]['capital']; ?></p>
    </body>

    </html>
```

Fecha y hora

PHP utiliza para las fechas y horas el formato de representación de Unix. Este formato representa una fecha y hora dada, por el número de segundos transcurridos desde el 1 de enero de 1970 00:00:00 GTM (Horas del meridiano de Greenwich). PHP dispone de numerosas funciones para trabajar con fechas y horas[19]. Veamos alguna de ellas, que son las que más vamos a utilizar.

Un problema que nos podemos encontrar es que nuestro servidor esté en una zona horaria distinta a la nuestra. Podríamos llevarnos la desagradable sorpresa de que la fecha o la hora no se correspondan con lo que esperábamos. Para evitar esto utilizaremos la función `date_default_timezone_set()`[20].

```
<?php
date_default_timezone_set('europe/madrid');

echo 'Timezone actual: ' . date_default_timezone_get();
?>
```

La función `time()`[21] nos devuelve la fecha y hora actual.

```
<?php
$time = time();

echo 'Hoy es: ' . date("d/m/Y", $time) . ', y son las ' . date("H:i", $time);
?>
```

La función `date()`[22] nos formatea la salida de la fecha y hora que le indiquemos. Le pasaremos como parámetro el formato de cómo queremos visualizar la fecha y hora. El segundo parámetro es opcional, si no se le indica nada, utilizará la fecha y hora actual.

Carácter de formato	Descripción
Día	
d	01 a 31
D	Lun a Dom
j	1 a 31
l	Lunes a Domingo
Semana	
W	Número de semana del año
Mes	
F	Enero a Diciembre
m	01 a 12
M	Ene a Dic
n	1 a 12

[19] http://php.net/manual/es/ref.datetime.php
[20] http://php.net/manual/es/function.date-default-timezone-set.php
[21] http://php.net/manual/es/function.time.php
[22] http://php.net/manual/es/function.date.php

t	Número de días del mes
Año	
L	1 si es año bisiesto, 0 en caso contrario
Y	Año de cuatro dígitos
y	Año de dos dígitos
Hora	
a	am o pm
A	AM o PM
g	1 a 12 (sin ceros delante)
h	01 a 12
G	0 a 24 (sin ceros delante)
H	00 a 24
i	00 a 59 (Minutos)
s	00 a 59 (Segundos)

Tabla 7. Formato de fecha y hora función `date()`

El siguiente ejemplo nos será de mucha ayuda más adelante, pues es el formato con el que guardaremos la fecha y hora en un campo de tipo `DateTime` de una tabla SQL.

```
<?php
$ahora           = time();
$formatoFechaSQL = date("Y-m-d H:m:s", $ahora);

echo $formatoFechaSQL;
?>
```

La siguiente función es `mktime()`[23], que la utilizaremos para crear una fecha en concreto. Le pasaremos como parámetros la hora, los minutos, los segundo, el mes, el día y el año, y nos devolverá la fecha y hora en formato UNIX

```
<?php
$fecha = mktime(2,20,37,10,12,1492);

echo date("Y-m-d G:i:s", $fecha);
?>
```

Por último, la función `getdate()`[24] que obtiene la fecha y hora del servidor y la guarda en un vector asociativo.

```
<?php
$ahora = getdate();

print_r($ahora);
?>
```

Superglobales

PHP proporciona una serie de variables por defecto que se llaman *superglobales* (*superglobals*[25]). Estas variables están disponibles a lo largo de nuestro programa.

[23] http://php.net/manual/es/function.mktime.php
[24] http://php.net/manual/es/function.getdate.php
[25] http://php.net/manual/es/language.variables.superglobals.php

Variable superglobal	Contenido
`$GLOBALS`	Vector asociativo con todas las variables globales de un programa.
`$_SERVER`	Vector con información del entorno del servidor y de ejecución.
`$_GET`	Vector asociativo pasado al programa vía parámetro URL.
`$_POST`	Vector asociativo pasado al programa vía método POST.
`$_FILES`	Vector asociativo con los archivos subidos vía método POST.
`$_COOKIE`	Vector asociativo con las variables pasadas al programa a través de cookies.
`$_SESSION`	Vector asociativo con los valores de la sesión.
`$_REQUEST`	Vector asociativo que contiene las variables `$_GET`, `$_POST` y `$_COOKIE`.
`$_ENV`	Vector asociativo con las variables del entorno.

Tabla 8 Variables superglobales

Veamos un ejemplo.

```
<?php
print r($ ENV);
?>
```

Constantes

En ocasiones querremos utilizar el mismo valor durante todo el programa. En este caso en lugar de utilizar una variable utilizaremos una *constante* (*constant*).

Las constantes utilizan la misma nomenclatura que las variables, pero no están precedidas por el símbolo `$`. Son sensibles a mayúsculas y minúsculas, pero por convenio las constantes son todas en mayúsculas. En lugar de utilizar el operador de asignación (=), utilizaremos la función `define()`[26].

```
<?php
define('SERVIDOR', 'localhost');
define('BASEDATOS', 'basedatos');
define('USUARIO', 'root');
define('CONTRASENYA', '12345');
?>
```

Si intentamos definir una constante previamente definida, nos dará un error.

Las constantes las utilizaremos, normalmente, para guardar valores de configuración.

Ejercicios

2. Haz un programa que visualice tu nombre y apellidos, ambos deben estar contenidos en variables.

3. Amplia el programa anterior, pero pidiendo el nombre y apellidos al usuario.

4. Pon nombre de variables coherentes a los siguientes conceptos.

[26] http://php.net/manual/es/function.define.php

- Piso en venta.
- Metros cuadrados.
- Valor de mercado.
- Nombre y apellidos.
- Examen de enero.
- Vacaciones de verano.
- Mes de cumpleaños.
- Alarma activa.
- Ubicación de un archivo.
- Dirección de una web.
- Nombre de la base de datos.
- Usuario de la base de datos.
- Contraseña de la base de datos.

5. Indica si el resultado de las siguientes expresiones es verdadero o falso, siendo $a = 6, $b = -5.8 y $c = 6.0.

    - $a < $b
    - $a >= $c
    - $b != $c
    - $a === $c
    - $a !== $c
    - $a / $b == 25
    - $a * $b == -34.8
    - ($a + $b) > ($b + $c)

6. Crea un programa que calcule la suma de los números pedidos al usuario, usando variables llamadas "numero1" y "numero2".

7. Crea un programa que calcule el producto de los números pedidos al usuario, usando variables llamadas "numero1" y "numero2".

8. Utilizando funciones de PHP localiza la posición de la primera y la última 'a' de la siguiente frase: "Alacant, la millor terreta del Mont".

9. Crea un vector asociativo que almacene los siguientes datos de tres compañeros de clase: Nombre, apellidos, teléfono y edad.

10. Crea un programa en PHP que muestre tu fecha de nacimiento en varios formatos.

Decisiones

Uno de los aspectos más importantes de un lenguaje de programación es la capacidad de poder tomar decisiones antes diferentes situaciones. Si conoces las sentencias de control, continúa con el siguiente capítulo, en caso contrario continúa leyendo.

if / else

La sentencia de control más conocida es el condicional *si* (*if*). Utilizaremos la sentencia `if` para decirle al programa que ejecute un código si una condición determinada es verdadera. Opcionalmente, podemos añadir una sentencia *si no* (*else*), para indicarle al programa que debe hacer si la condición es falsa.

A continuación, se muestra la sintaxis de la sentencia `if/else`.

Sintaxis if/else	Sintaxis alternativa

`if (condición)`
`{`
`  Líneas código 1;`
`  ...`
`}`
`else`
`{`
`  Líneas código 2;`
`  ...`
`}`
	`if (condición) :`
	`  Líneas código 1;`
	`  ...`
	`else :`
	`  Líneas código 2;`
	`  ...`
	`endif;`

Veamos unos ejemplos.

```
<?php
$nombre = 'Javier';

if ($nombre != 'Javier')
    echo 'Yo no me llamo Javier';
else
    echo 'Yo soy Javier';

echo '<br/>Ahora con la sentencia alternativa<br/>';

if ($nombre != 'Javier') :
    echo 'Yo no me llamo Javier';
else :
    echo 'Yo soy Javier';
endif;
?>
```

Cuando las sentencias a ejecutar dentro de las condiciones sean múltiples las pondremos entre llaves de conjunto.

```
<?php
$nombre    = 'Javier';
$apellidos = 'Miras';

if ($nombre != 'Javier')
{
    echo 'Yo no me llamo Javier';

    if ($apellidos == 'Pérez García')
        echo " y me apellido $apellidos";
}
else
    echo "Yo soy Javier $apellidos";
```

```
echo '<br/>Ahora con la sentencia alternativa<br/>';

if ($nombre != 'Javier') :
    echo 'Yo no me llamo Javier';

    if ($apellidos == 'Pérez García') :
        echo " y me apellido $apellidos";
    endif;
else :
    echo 'Yo soy Javier';
endif;

?>
```

Cuando se hacen comparaciones de igualdad (==) dentro de la sentencia `if`, nos puede ocurrir que se nos olvide poner los dos signos de igual. Esto hace que, en lugar de realizarse la comparación se haga una asignación. Para evitar esto es aconsejable, cuando se pueda hacer, poner primero la constante y después la variable. Si se nos olvida un igual, PHP no dará un error.

```
<?php
$nombre    = 'Javier';
$apellidos = 'Miras Llamas';

if ('Javier' != $nombre)
{
    echo 'Yo no me llamo Javier';

    if ('Pérez García' = $apellidos)  // Nos dará un error por olvidarnos de un '='.
        echo " y me apellido $apellidos";
}
else
    echo "Yo soy Javier $apellidos";
?>
```

Cuando tengamos que analizar más de una condición hay que recordar el orden de precedencia de los operadores condicionales, por lo que, es aconsejable poner paréntesis para que el resultado sea el esperado.

elseif

En ocasiones utilizaremos repetidamente la sentencia condicional `if`, anidándola con la sentencia `else`. Para estos casos PHP nos proporciona la sentencia `elseif`.

A continuación, se muestra la sintaxis de la sentencia `elseif`.

Sintaxis elseif	Sintaxis alternativa

`if (condición 1)`
`{`
`  Líneas código 1;`
`  ...`
`}`
`elseif (condición 2)`
`{`
`  Líneas código 2;`
`  ...`
`}`
`else`
`{`
`  Líneas código 3;`
`  ...`
`}`
	`if (condición 1) :`
	`  Líneas código 1;`
	`  ...`
	`elseif (condición 2) :`
	`  Líneas código 2;`
	`  ...`
	`else :`
	`  Líneas código 3;`
	`  ...`
	`endif;`

Veamos un ejemplo.

```
<?php
$diaSemana = 'Jueves';
```

```
if ('Lunes' == $diaSemana)
    echo 'Hoy es lunes.';
elseif ('Martes' == $diaSemana)
    echo 'Hoy es martes.';
elseif ('Miércoles' == $diaSemana)
    echo 'Hoy es miércoles.';
elseif ('Jueves' == $diaSemana)
    echo 'Hoy es jueves.';
elseif ('Viernes' == $diaSemana)
    echo 'Hoy es viernes.';
elseif (('Sábado' == $diaSemana) || ('Domingo' == $diaSemana))
    echo 'Hoy es fin de semana.';
else
    echo 'Hoy no tengo ni idea que día es.';
?>
```

Operador ternario

El operador ternario es una sentencia `if/else` escrita de forma abreviada. Su sintaxis es la siguiente.

Sintaxis operador ternario

```
((condición) ? a hacer si es verdadera : a hacer si es falsa);
```

Veamos un ejemplo.

```
<?php
$casado = true;

echo ((true == $casado) ? 'Estoy felizmente casado.' : 'Tengo que solucionar esto.');
?>
```

La única limitación que tenemos con el operador ternario es que las sentencias a realizar tienen que ser cortas y una sola, no se permite más de una sentencia.

```
<?php
$casado = true;

echo 'Casado: ' . ((true == $casado) ? 'Sí' : 'No');
?>
```

Recordad este último código, nos será de utilidad.

Operador coalescente NULL

En PHP, a partir de la versión 7, disponemos del operador de coalescente NULL (`??`). Este operador tiene dos términos, devolverá el primero si este existe, en caso contrario devuelve el segundo.

```
<?php

echo $casado ?? 'no se sabe'; // echo (isset($casado) ? $casado : 'no se sabe');

?>
```

switch

La sentencia de *selección múltiple* (*swtich*), la utilizaremos cuando queramos comparar una variable o expresión con diferentes valores simples. La sintaxis del es la siguiente:

Sintaxis switch	Sintaxis alternativa

`switch (condición)`
`{`
`  case opción 1:`
`  Líneas código 1;`
`  ...`
`  break;`
`  case opción 2:`
`  Líneas código 2;`
`  ...`
`  break;`
`  default:`
`  Líneas código defecto;`
`  ...`
`  break;`
 	
`}`
	`switch (condición) :`
	`  case opción 1:`
	`  Líneas código 1;`
	`  ...`
	`  break;`
	`  case opción 2:`
	`  Líneas código 2;`
	`  ...`
	`  break;`
	`  default:`
	`  Líneas código defecto;`
	`  ...`
	`  break;`
	`endswith;`

Veamos un ejemplo.

```
<?php
$diaSemana = 'Sábado';

switch ($diaSemana)
{
    case 'Lunes':
        echo 'Hoy es lunes.';
        break;
    case 'Martes':
        echo 'Hoy es martes.';
        break;
    case 'Miércoles':
        echo 'Hoy es miércoles.';
        break;
    case 'Jueves':
        echo 'Hoy es jueves.';
        break;
    case 'Viernes';
        echo 'Hoy es viernes.';
        break;
    case 'Sábado':
    case 'Domingo':
        echo 'Hoy es fin de semana.';
        break;
    default:
        echo 'Hoy no tengo ni idea que día es.';
}
?>
```

Cuidado de no olvidar poner un `break` cuando queramos finalizar una opción, pues en caso contrario entraría en la siguiente.

Ejercicios

11. Un programa que pida al usuario una frase, después una letra y finalmente diga si aparece esa letra como parte de esa frase o no. Utilizar una función del API de PHP.

12. Crea un programa que pida al usuario un número entero y diga si es par.

13. Crea un programa que pida al usuario dos números enteros y diga cuál es el mayor de ellos.

14. Crea un programa que pida al usuario dos números enteros y diga si el primero es múltiplo del segundo.

15. Crea un programa que pida al usuario dos números entero. Si el primer número es múltiplo de 10, informará al usuario e indicará si el segundo número también es múltiplo de 10.

16. Crea un programa que multiplique dos números pedidos al usuario. Si cualquiera de los números es cero, deberá aparecer en pantalla: "El producto por 0 es 0".

17. Crea un programa que pida al usuario dos números enteros. Si el segundo no es cero, mostrará el resultado de dividir entre el primero y el segundo. En caso contrario, escribirá "Error: No se puede dividir entre cero"

18. Crea un programa que pida al usuario un número entero y responda si es múltiplo de 2 o de 3.

19. Crea un programa que pida al usuario un número entero y responda si es múltiplo de 2 y de 3 simultáneamente.

20. Crea un programa que pida al usuario un número entero y responda si es múltiplo de 2 pero no de 3.

21. Crea un programa que pida al usuario un número entero y responda si no es múltiplo de 2 ni de 3.

22. Crea un programa que pida al usuario dos números enteros y diga si ambos son pares.

23. Crea un programa que pida al usuario dos números enteros y diga si (al menos) uno es par.

24. Crea un programa que pida al usuario dos números enteros y diga si uno y sólo uno es par.

25. Crea un programa que pida al usuario dos números enteros y diga "Uno de los números es positivo", "Los dos números son positivos" o bien "Ninguno de los números es positivo", según corresponda.

26. Crea un programa que pida al usuario tres números y muestre cuál es el mayor de los tres.

27. Crea un programa que pida al usuario dos números enteros y diga si son iguales o, en caso contrario, cuál es el mayor de ellos.

28. Crea un programa que use el operador condicional para mostrar un el valor absoluto de un número de la siguiente forma: si el número es positivo, se mostrará tal cual; si es negativo, se mostrará cambiado de signo.

29. Usa el operador condicional para calcular el menor de dos números.

30. Crea un programa que pida un número del 1 al 5 al usuario, y escriba el nombre de ese número, usando "switch" (por ejemplo, si introduce "1", el programa escribirá "uno").

31. Crea un programa que pida una letra tecleada por el usuario y diga si se trata de un signo de puntuación (. , ; :), una cifra numérica (del 0 al 9) o algún otro carácter, usando "switch".

32. Crea un programa que pida una letra tecleada por el usuario y diga si se trata de una vocal, una cifra numérica o una consonante, usando "switch".

33. Repite el ejercicio 30, empleando "if" en lugar de "switch".

34. Repite el ejercicio 31, empleando "if" en lugar de "switch" (pista: como las cfras numéricas del 0 al 9 están ordenadas, no hace falta comprobar los 10 valores, sino que se puede hacer con "if ((simbolo >= '0') && (simbolo <='9'))").

35. Repite el ejercicio 32, empleando "if" en lugar de "switch".

Bucles

Los bucles nos permiten repetir bloques de código varias veces. Dependiendo del tipo de bucle, podremos repetir el código un número determinado de veces o mientras sea verdadero una cierta condición.

También aprenderemos como salir de un bucle o ir a la próxima repetición utilizando `break` y `continue`.

while

Los bucles *mientras* (*while*[27]) ejecutan un bloque mientras una condición dada sea verdadera. Esta es la sintaxis de un bucle mientras.

Sintaxis while	Sintaxis alternativa

`while (condición salida)`
`{`
`  Líneas código;`
`  ...`
`}`
	`while (condición salida) :`
	`  Líneas código;`
	`  ...`
	`endwhile;`

En este tipo de bucle es común establecer un contador y repetir el bucle hasta que se alcanza una condición. Cada repetición del bucle se llama *iteración* (*iteration)*.

```
<?php
$i = 1;

while ($i < 5)
{
    echo "<p>El contador vale $i.</p>";
    $i++;
}
?>
```

El programa anterior mostrará en pantalla cuatro veces la frase "El contador vale ...", desde uno hasta 4.

Ojo con la condición de salida del bucle. No es lo mismo que la condición sea estrictamente menor que cinco, que menor o igual que cinco. En el primer caso se repite cuatro veces, en el segundo cinco.

Otro error que se suele cometer a veces es olvidar se incrementar el contador. La condición de salida del bucle nunca se daría y nos meteríamos en un bucle infinito.

do/while

Los bucles *mientras hacer* (*do/while*[28]) son como los bucles `while`, con la diferencia de que la condición de salida del bucle se evalúa una vez se hay hecho una primera iteración. En bucle `while` si la condición es falsa no se entre en él, mientras que en un bucle `do/while` se iterará una

[27] http://php.net/manual/es/control-structures.while.php
[28] http://php.net/manual/es/control-structures.do.while.php

vez. A continuación, viene la sintaxis para un bucle `do/while`, este tipo de bucle no tiene sintaxis alternativa.

Sintaxis do/while	Sintaxis alternativa

`do`
`{`
`  Líneas código;`
`  ....`
`} while (condición salida);`
	No existe.

Veamos el ejemplo anterior con un bucle `do/while`.

```
<?php
$i = 1;

do
{
    echo "<p>El contador vale $i.</p>";
    $i++;
} while ($i < 5);
?>
```

for

Los bucles *para* (*for*[29]) repiten un bloque de código un número determinado de veces. La sintaxis de un bucle `for` es la siguiente.

Sintaxis for	Sintaxis alternativa

`for (expre1; expre2; expre3)`
`{`
`  Líneas código;`
`  ...`
`}`
	`for (expre1; expre2; expre3) :`
	`  Líneas código;`
	`  ...`
	`endfor;`

La primera `expre1` del bucle `for` se evalúa al comienzo del mismo. A continuación, y cada iteración, se evalúa la `expre2`, si es verdadera el bucle continúa, es caso contrario finaliza. La `expre3` se evalúa al final de cada iteración y sirve para incrementar el contador.

```
<?php
for ($i = 0; $i < 5; $i++)
{
    echo "<p>El contador vale $i.</p>";
}
?>
```

El bucle iterará cinco veces mostrando "El contador vale ..." desde cero hasta cuatro.

Cuidado con la inicialización del contador y la condición de finalización.

Los bucles `for` son muy útiles para trabajar con vectores.

```
<?php
$comunidadValenciana =
    array(array('provincia' => 'Alicante', 'capital' => 'Alicante'),
          array('provincia' => 'Castellón', 'capital' => 'Castellón de la plana'),
          array('provincia' => 'Valencia', 'capital' => 'Valencia'));

$fin = count($comunidadValenciana);

for ($i = 0; $i < $fin; $i++)
```

[29] http://php.net/manual/es/control-structures.for.php

```
{
    echo "<p>Provincia: {$comunidadValenciana[$i]['provincia']}</p>";
    echo "<p>Capital: {$comunidadValenciana[$i]['capital']}</p>";
}
?>
```

Observa que el vector `$comunidadValenciana` está puesto entre llaves de conjunto (`{}`) para poder mostrarlo, en caso contrario nos daría un error.

foreach

Los bucles *para cada* (*foreach*[30]) es una forma útil de trabajar con vectores. Su sintaxis es la siguiente.

Sintaxis foreach	Sintaxis alternativa

`foreach ($vector as $elemento)`
`{`
`  Líneas código $elemento;`
`  ...`
`}`
	`foreach ($vector as $elemento) :`
	`  Líneas código $elemento;`
	`  ...`
	`endforeach;`

Veamos un ejemplo.

```
<?php
$comunidadValenciana =
       array(array('provincia' => 'Alicante', 'capital' => 'Alicante'),
       array('provincia' => 'Castellón', 'capital' => 'Castellón de la plana'),
       array('provincia' => 'Valencia', 'capital' => 'Valencia'));

foreach ($comunidadValenciana as $provincia)
{
    echo "<p>Provincia: {$provincia['provincia']}</p>";
    echo "<p>Capital: {$provincia['capital']}</p>";
}
?>
```

Cuando el vector sea asociativo podemos obtener el índice. Para ello utilizaremos la siguiente sintaxis.

Sintaxis	Sintaxis alternativa

`foreach ($vector as $clave => $elemento)`
`{`
`  Líneas código $clave/$elemento;`
`  ...`
`}`
	`foreach ($vector as $clave => $elemento) :`
	`  Líneas código $clave/$elemento;`
	`  ...`
	`endforeach;`

Veamos un ejemplo.

```
<?php
$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');

foreach ($usuario as $clave => $valor)
{
    echo "<p>" . ucfirst($clave) . " = $valor </p>";
}
?>
```

[30] http://php.net/manual/es/control-structures.foreach.php

La función `ucfirst()`[31] convierte el primer carácter de una cadena en mayúsculas.

continue

La palabra reservada *continuar* (*continue*[32]) la utilizaremos cuando queramos saltarnos el código que hay a continuación y hacer la siguiente iteración.

```
<?php
$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');


foreach ($usuario as $clave => $valor)
{
    if ($clave == 'casado')
        continue;

    echo "<p>" . ucfirst($clave) . " = $valor </p>";
}
?>
```

En este caso no mostraremos la condición de `casado`. Al comparar la `$clave`, y comprobar que es `casado`, el `continue` hará que vayamos a la siguiente iteración.

break

La palabra reservada *romper* (*break*[33]) la utilizaremos para salirnos completamente del bucle.

```
<?php
$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');


foreach ($usuario as $clave => $valor)
{
    if ($clave == 'casado')
        break;

    echo "<p>" . ucfirst($clave) . " = $valor </p>";
}
?>
```

Sólo mostrará `nombre` y `apellidos`, cuando llegue a comprobar que la `$clave` es `casado`, saldrá del bucle y no llegará a imprimir `fechanacimiento`.

La utilización de `continue` o `break` puede hacer nuestro código algo ilegible, nos puede llevar a cometer errores y puede ser más difícil de depurar. Un simple `continue` o `break` no es problema, como en los ejemplos anteriores, pero la utilización de más de uno en el mismo bloque de código es desaconsejable.

[31] http://php.net/manual/es/function.ucfirst.php
[32] http://php.net/manual/es/control-structures.continue.php
[33] http://php.net/manual/es/control-structures.break.php

Ejercicios

36. Crea un programa que escriba en pantalla los números del 1 al 10, usando "while".

37. Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "while".

38. Crea un programa calcule cuantas cifras tiene un número entero positivo usando "while" (Se puede hacer dividiendo varias veces entre 10).

39. Crea un programa que escriba en pantalla los números del 1 al 10, usando "do..while".

40. Crea un programa que escriba en pantalla los números pares del 26 al 10 (descendiendo), usando "do..while".

41. Crea un programa que muestre los números del 10 al 20, ambos incluidos, usando "for".

42. Crea un programa que escriba en pantalla los números del 1 al 50 que sean múltiplos de 3 (habrá que recorrer todos esos números y ver si el resto de la división entre 3 resulta 0), usando "for".

43. Crea un programa que muestre los números del 100 al 200 (ambos incluidos) que sean divisibles entre 7 y a la vez entre 3, usando "for".

44. Crea un programa que muestre la tabla de multiplicar del 9, , usando "for".

45. Crea un programa que muestre los primeros ocho números pares: 2 4 6 8 10 12 14 16 (en cada pasada habrá que aumentar de 2 en 2, o bien mostrar el doble del valor que hace de contador), usando "for".

46. Crea un programa que muestre los números del 15 al 5, descendiendo (en cada pasada habrá que descontar 1, por ejemplo, haciendo i=i-1, que se puede abreviar i--), usando "for".

47. Crea un programa escriba 4 veces los números del 1 al 5, en una misma línea, usando "for": 12345123451234512345.

48. Crea un programa escriba 4 veces los números del 1 al 5, en una misma línea, usando "while": 12345123451234512345.

49. Crea un programa que, para los números entre el 10 y el 20 (ambos incluidos) diga si son divisibles entre 5, si son divisibles entre 6 y si son divisibles entre 7.

50. Crea un programa que escriba 4 líneas de texto, cada una de las cuales estará formada por los números del 1 al 5.

51. Crea un programa que pida al usuario dos números y escriba su máximo común divisor (pista: una solución lenta pero sencilla es probar con un "for" todos los números descendiendo a partir del menor de ambos, hasta llegar a 1; cuando encuentres un número que sea divisor de ambos, interrumpes la búsqueda).

52. Crea un programa que pida al usuario dos números y escriba su mínimo común múltiplo (pista: una solución lenta pero sencilla es probar con un "for" todos los números a partir del mayor de ambos, de forma creciente; cuando encuentres un número que sea múltiplo de ambos, interrumpes la búsqueda).

53. Crea un programa que escriba los números del 20 al 10, descendiendo, excepto el 13, usando "continue".

54. Crea un programa que escriba los números pares del 2 al 106, excepto los que sean múltiplos de 10, usando "continue"

55. Crea un programa que escriba los números pares del 20 al 10, descendiendo, excepto el 14, primero con "for" y luego con "while".

56. Crea un programa que calcule un número elevado a otro, usando multiplicaciones sucesivas.

57. Crea un programa que "dibuje" un rectángulo formado por asteriscos, con el ancho y el alto que indique el usuario, usando dos "for" anidados. Por ejemplo, si desea anchura 4 y altura 3, el rectángulo sería así:

```
****
****
****
```

58. Crea un programa que "dibuje" un triángulo decreciente, con la altura que indique el usuario. Por ejemplo, si el usuario dice que desea 4 caracteres de alto, el triángulo sería así:

```
****
***
**
*
```

59. Crea un programa que "dibuje" un rectángulo hueco, cuyo borde sea una fila (o columna) de asteriscos y cuyo interior esté formado por espacios en blanco, con el ancho y el alto que indique el usuario. Por ejemplo, si desea anchura 4 y altura 3, el rectángulo sería así:

```
****
*  *
****
```

60. Crea un programa que "dibuje" un triángulo creciente, alineado a la derecha, con la altura que indique el usuario. Por ejemplo, si el usuario dice que desea 4 caracteres de alto, el triángulo sería así:

```
   *
  **
 ***
****
```

61. Crea un programa que devuelva el cambio de una compra, utilizando monedas (o billetes) del mayor valor posible. Supondremos que tenemos una cantidad ilimitada de monedas (o billetes) de 100, 50, 20, 10, 5, 2 y 1, y que no hay decimales. La ejecución podría ser algo como:

```
¿Precio? 44
¿Pagado? 100

Su cambio es de 56: 50 5 1

¿Precio? 1
¿Pagado? 100

Su cambio es de 99: 50 20 20 5 2 2
```

Funciones y procedimientos

El escribir código conlleva consumir tiempo, probarlo, depurarlo y, aun así, puede contener errores. Cuando escribimos código, y este funciona correctamente, podemos reutilizarlo copiando y pegándolo, en lugar de volver a escribirlo. Pero nos puede llevar a contener errores que tendremos que corregir; por ejemplo, renombrar las variables.

Para evitar esto utilizaremos las *funciones* (*functions*) y los *procedimientos* (*procedures*). La única diferencia entre una función y un procedimiento es que la primera devuelve un valor, mientras que el segundo no devuelve nada.

Ya hemos vista hasta ahora varias funciones que nos proporciona PHP[34]: `echo()`, `print_r()`, `unset()`, `is_null()`, `date_default_timezone_set()`, `time()`, `ucfirst()`, etc. Cuando necesitemos algo, miraremos primero si PHP nos proporciona una función o procedimiento.

Como las operativa de las funciones y procedimientos es la misma, nos centraremos en las funciones.

Funciones

El nombre de las funciones sigue la misma regla que las variables, excepto que no comienzan por el símbolo del dólar (`$`) y justo a continuación lleva paréntesis. La sintaxis es la siguiente.

Sintaxis de una función

```
function nombreFuncion(parametro1,
                       &parametro2,
                       ...
                       parametroN = valor por defecto)
{
    Líneas de código;
    ....

    return valor;
}
```

Para devolver un valor utilizaremos la palabra reservada `return`. Podemos utilizarla en cualquier parte del código de nuestra función, haciendo que finalice la ejecución de esta.

Veamos un ejemplo.

```
<?php
function obtenerUsuario()
{
    $cadena  = "";
    $usuario = array('nombre' => 'Javier',
                     'apellidos' => 'Miras',
                     'casado' => 'Sí',
                     'fechanacimiento' => '08/02/....');

    foreach ($usuario as $clave => $valor)
    {
        $cadena = $cadena . "<p>" . ucfirst($clave) . " = $valor </p>";
    }

    return $cadena;
}
```

[34] http://php.net/manual/es/indexes.functions.php

```
echo obtenerUsuario();
?>
```

Veamos el anterior ejemplo utilizando un procedimiento.

```
<?php
function imprimirUsuario()
{
    $cadena  = "";
    $usuario = array('nombre' => 'Javier',
                     'apellidos' => 'Miras',
                     'casado' => 'Sí',
                     'fechanacimiento' => '08/02/....');

    foreach ($usuario as $clave => $valor)
    {
        $cadena = $cadena . "<p>" . ucfirst($clave) . " = $valor </p>";
    }

    echo $cadena;
}

imprimirUsuario();
?>
```

El resultado es el mismo. Pero qué pasa si el usuario no quiero mostrarlo por pantalla. Mi experiencia me dice que es mejor la primera opción, devolver la cadena resultante y después ya veremos qué hacemos con ella.

Declaración de tipo de retorno

Desde la versión 7 se puede definir el tipo devuelto por una función La sintaxis es la siguiente.

Sintaxis de una función

```
function nombreFuncion(parametro1,
                       &parametro2,
                       ...
                       parametroN = valor por defecto) : tipo
{
    Líneas de código;
    ....

    return valor;
}
```

De esta forma no es necesario hacer la conversión automática del valor devuelto por la función, puesto que ya sabemos que valor va a devolver.

```
<?php

function Suma($a, $b) : int
{
    return $a + $b;
}

echo Suma(5, 10);
?>
```

Parámetros

Un inconveniente de la función anterior es que siempre muestra el mismo resultado. Para que la función muestre distintos resultados utilizaremos los *parámetros* (*parameters*). Los parámetros son las variables que le pasamos a la función dentro de los paréntesis.

```
<?php
```

```
$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');

function obtenerUsuario($usuario)
{
    $cadena = '';

    foreach ($usuario as $clave => $valor)
    {
        $cadena = $cadena . "<p>" . ucfirst($clave) . " = $valor </p>";
    }

    return $cadena;

}

echo obtenerUsuario($usuario);
?>
```

En el ejemplo anterior tenemos un parámetro llamado `$usuario` al cual le pasamos como *argumento* (*argument*) un vector llamado `$usuario`. Fijaros que, aunque se llaman igual, no son la misma cosa. Esta técnica no es la más apropiada, pero es un buen ejemplo para que se entienda la diferencia entre uno y otro.

```
<?php

$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');

$usuario2 = array('nombre' => 'José',
                 'apellidos' => 'García',
                 'casado' => 'No',
                 'fechanacimiento' => '14/11/....');

function obtenerUsuario($usuario)
{
    $cadena = '';

    foreach ($usuario as $clave => $valor)
    {
        $cadena = $cadena . "<p>" . ucfirst($clave) . " = $valor </p>";
    }

    return $cadena;

}

echo obtenerUsuario($usuario);
echo obtenerUsuario($usuario2);
?>
```

Parámetros por valor

Esta forma de pasar los argumentos a la función se denomina *parámetros por valor*. Aunque cambiemos el argumento pasado a la función una vez que este sale de la misma recupera el valor inicial.

```
<?php
$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');

function parametroPorValor($usuario)
{
    $usuario['nombre']    = 'José';
    $usuario['apellidos'] = 'García';
    $usuario['casado']    = 'No';
```

```
    $usuario['apellidos'] = '14/11/....';
}

function obtenerUsuario($usuario)
{
    $cadena = '';

    foreach ($usuario as $clave => $valor)
    {
        $cadena = $cadena . "<p>" . ucfirst($clave) . " = $valor </p>";
    }

    return $cadena;

}

parametroPorValor($usuario);

echo obtenerUsuario($usuario);
?>
```

Parámetros por referencia

Otra forma de pasar los argumentos a la función se denomina *parámetros por referencia*. En este caso si cambiamos el argumento pasado a la función una vez que este sale de la misma no recupera el valor inicial.

```
<?php
$usuario = array('nombre' => 'Javier',
                 'apellidos' => 'Miras',
                 'casado' => 'Sí',
                 'fechanacimiento' => '08/02/....');

function parametroPorReferencia(&$usuario)
{
    $usuario['nombre']    = 'José';
    $usuario['apellidos'] = 'García';
    $usuario['casado']    = 'No';
    $usuario['apellidos'] = '14/11/....';
}

function obtenerUsuario($usuario)
{
    $cadena = '';

    foreach ($usuario as $clave => $valor)
    {
        $cadena = $cadena . "<p>" . ucfirst($clave) . " = $valor </p>";
    }

    return $cadena;

}

parametroPorReferencia($usuario);

echo obtenerUsuario($usuario);
?>
```

Observa que la única diferencia entre la función `parametroPorValor($usuario)` y `parametroPorReferencia(&$usuario)`, aparte del nombre, es el signo *ampersand* (`&`) que lleva el parámetro `$usuario`.

Parámetros por defecto

Los parámetros por defecto son aquellos que contienen un argumento predefinido.

```
<?php
function saludar($nombre = 'desconocido')
{
    return "Hola $nombre";
```

```
}

echo '<p>' . saludar() . '</p>';
echo '<p>' . saludar('Javier') . '</p>';
?>
```

Declaración de tipo de parámetro

Desde la versión 5 se puede definir el tipo de los parámetros que se le pasan a una función La sintaxis es la siguiente.

Sintaxis de una función

```
function nombreFuncion(tipo parametro1,
                       tipo &parametro2,
                       ...
                       tipo parametroN = valor por defecto)
{
    Líneas de código;
    ....

    return valor;
}
```

En el caso de que no pueda hacer la conversión se producirá una excepción `TypeError`.

```
<?php

function Suma(int $a, int $b) : int
{
    return $a + $b;
}

$a = 5.5;
$b = 3;

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

Si queremos que nuestras funciones sean estrictas en cuanto al tipo de los parámetros utilizaremos la declaración `declare(strict_types=1);` al comienzo de cada script.

```
<?php

declare(strict_types=1); // Si no se pasa el tipo correcto se produce
                         // la excepción TypeError.

function Suma(int $a, int $b) : int
{
    return $a + $b;
}

$a = 5.5; // No dará una excepción TypeError, porque no es de tipo entero.
$b = 3;

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

Ámbito de las variables

El *ámbito* (*scope*[35]) de una variable es el contexto donde se ha definido. En la función `obtenerUsuario()` hemos definido una variable `$cadena` dentro de ella, cuyo ámbito de existencia termina cuando finaliza la ejecución de esta.

[35] http://php.net/manual/es/language.variables.scope.php

global

Si hemos definido una variable fuera de una función y queremos utilizarla dentro de la función debemos declararla como global o utilizar la variable reservada `$GLOBALS`[36].

Veamos un ejemplo.

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

function sumar()
{
    global $b;

    $b = $GLOBALS['a'] + $b;
}

sumar();

echo $b;
?>
```

static

Una variable *estática* (*static*[37]) es aquella que se declara en una función, pero que no pierde su valor cuando finaliza la ejecución de esta.

```
<?php
$a = 1;

function sumar()
{
    global $a;
    static $b = 0;

    $b = $a + $b;

    return $b;
}

echo '<p>' . sumar() . '</p>';
echo '<p>' . sumar() . '</p>';
echo '<p>' . sumar() . '</p>';
?>
```

La variable `$b` mantendrá su valor, aunque finalice la ejecución de la función. Este tipo de variables es muy útil para generar contadores.

Ejercicios

62. Crea una función llamada "DibujarCuadrado3x3", que dibuje un cuadrado formato por 3 filas con 3 asteriscos cada una.

63. Crea una función "DibujarCuadrado" que dibuje en pantalla un cuadrado del ancho (y alto) que se indique como parámetro.

64. Crea una función "DibujarRectangulo" que dibuje en pantalla un rectángulo del ancho y alto que se indiquen como parámetros.

[36] http://php.net/manual/es/reserved.variables.globals.php
[37] http://php.net/manual/es/language.variables.scope.php#language.variables.scope.static

65. Crea una función "Cubo" que calcule el cubo de un número real (float) que se indique como parámetro. El resultado deberá ser otro número real. Prueba esta función para calcular el cubo de 3.2 y el de 5.

66. Crea una función "Menor" que calcule el menor de dos números enteros que recibirá como parámetros. El resultado será otro número entero.

67. Crea una función llamada "Signo", que reciba un número real, y devuelva un número entero con el valor: -1 si el número es negativo, 1 si es positivo o 0 si es cero.

68. Crea una función "Inicial", que devuelva la primera letra de una cadena de texto. Prueba esta función para calcular la primera letra de la frase "Hola".

69. Crea una función "UltimaLetra", que devuelva la última letra de una cadena de texto. Prueba esta función para calcular la última letra de la frase "Hola".

70. Crea una función "MostrarPerimSuperfCuadrado" que reciba un número y calcule y muestre en pantalla el valor del perímetro y de la superficie de un cuadrado que tenga como lado el número que se ha indicado como parámetro.

71. Crea una función "EscribirTablaMultiplicar" que escriba la tabla de multiplicar de un número que se declarará variable global (por ejemplo, para el 3 deberá llegar desde "3x0=0" hasta "3x10=30").

72. Crea una función "ContarLetra", que reciba como parámetro una letra, y devuelva la cantidad de veces que dicha letra aparece en la cadena declarada como global. Por ejemplo, si la cadena es "Alicante" y la letra es 'a', debería devolver 2.

73. Crea una función "Intercambiar", que intercambie el valor de los dos números que se le indiquen como parámetro, usando parámetros por referencia.

74. Crea una función "Iniciales", que reciba una cadena como "IG Formación" y devuelva las letras I y F (primera letra, y letra situada tras el primer espacio), usando parámetros por referencia.

75. Crea una función que imite el lanzamiento de un dado, generando un número al azar entre 1 y 6.

76. Crea una función que genere un array relleno con 100 números reales al azar entre -1000 y 1000. Luego deberá calcular, y mostrar e pantalla, la media y la mediana.

77. Haz un programa que resuelva ecuaciones de segundo grado, del tipo $a \cdot x^2 + b \cdot x + c = 0$. El usuario deberá introducir los valores de a, b y c. Se deberá crear una función "CalcularRaicesSegundoGrado", que recibirá como parámetros los coeficientes a, b y c (por valor), así como las soluciones x_1 y x_2 (por referencia). Deberá devolver los valores de las dos soluciones x_1 y x_2. Si alguna solución no existe, se devolverá como valor 100.000 para esa solución.

78. Crea un programa que pida al usuario un ángulo (en grados) y devuelva en tres parámetros el seno, el coseno y la tangente. Recuerda que las funciones trigonométricas esperan que el ángulo se indique en radianes, no en grados. La equivalencia es que 360 grados son 2*PI radianes.

79. La empresa para la que trabajas quiere que realices un informe sobre las funciones anónimas o closures.

Inclusión y redirección de archivos

Conforme vayamos desarrollando nuestro programa este irá creciendo en líneas de código. En ocasiones será casi imposible seguir, comprender y organizar nuestro código. Por otro lado, si todo está incluido en un único fichero será imposible aprovechar parte de nuestro código en otros proyectos.

Hemos visto que las funciones nos ayudan a reutilizar nuestro código. Si juntamos varias funciones que realizan tareas similares, iremos creando nuestras propias bibliotecas. Cuando queramos utilizarlas tendremos que incluirlas en nuestro programa.

include / include_once

Mediante la función de *incluir* (*include*[38]) se incluye y evalúa un archivo. Su sintaxis es la siguiente.

```
Sintaxis de include
```

```
include(urlDelArchivoAIncluir);
```

Si el fichero que queremos incluir se utiliza en varios sitios de nuestro programa, utilizaremos la función de *incluir una vez* (*include_once*[39]). De esta forma, sólo se incluirá una vez el archivo. Su sintaxis es la siguiente.

```
Sintaxis de include_once
```

```
include_once(urlDelArchivoAIncluirUnoSolaVez);
```

Veamos un ejemplo.

```
// usuarios.php

<?php
$nombre    = 'Javier';
$apellidos = 'Miras';
?>

// mostrarusuarios.php

<?php
include('usuarios.php');

echo "Mi nombre es $nombre $apellidos";
?>
```

require / require_once

En ninguno de los dos casos, tanto si utilizo `include` como `include_once`, la ejecución de nuestro programa no finaliza. Si fuese necesario que nuestro programa finalizase en el caso de no poder cargar el fichero, utilizaríamos la función de *requerir* (*require*[40]). Su sintaxis es la siguiente.

```
Sintaxis de require
```

[38] http://php.net/manual/es/function.include.php
[39] http://php.net/manual/es/function.include-once.php
[40] http://php.net/manual/es/function.require.php

```
require(urlDelArchivoAIncluir);
```

En el caso de que sólo sea necesario requerir una sola vez un fichero, utilizaremos la función de *requerir una vez* (*require_once*[41])

Sintaxis de include_once

```
require_once(urlDelArchivoAIncluirUnoSolaVez);
```

Veamos un ejemplo.

```
// usuarios.php

<?php
$nombre    = 'Javier';
$apellidos = 'Miras';
?>
```

```
// mostrarusuarios.php

<?php
require('usuarios.php');

echo "Mi nombre es $nombre $apellidos";
?>
```

header

Cuando queramos redirigir nuestro script a otro utilizaremos la función `header`[42], la cual envía encabezados HTTP sin formato.

Su sintaxis es la siguiente.

Sintaxis de header

```
header(string $string, bool $replace = true, int $http_response_code = ?): void
```

Normalmente, utilizaremos `header` para redirigir el script y es importante que después de la función se utilice `exit`[43] (o su equivalente `die)` para parar la ejecución de este.

Veamos un ejemplo.

```
<?php
header('Location: https://www.dominio.com/otroscript.php');

exit;
?>
```

Ejercicios

80. Crea un fichero, que se llamará "funciones.php", en el contendrá todas las funciones que se han creado en ejercicios anteriores. Haz un programa que incluya es fichero y prueba alguna de estas funciones.

[41] http://php.net/manual/es/function.require-once.php
[42] http://php.net/manual/es/function.header.php
[43] http://php.net/manual/es/function.exit.php

Errores

Los errores pueden producirse de diferente manera y tipo. El primer tipo, con el que te vas a familiariza rápidamente, son los errores de programación. Estos errores se producen por un uso incorrecto de la sintaxis. El segundo es son las excepciones, que son aquellos errores que se producen en tiempo de ejecución.

Si tenemos activado `display_errors`[44] en nuestro php.ini, PHP nos mostrará la información acerca del error producido. Como norma general, debemos mostrar todos los errores mientras estemos desarrollando nuestro programa y los desactivaremos cuando lo pongamos en producción.

Parse

Un `Parse` se produce cuando el analizador de PHP se encuentra con un error sintáctico. Este tipo de error supone la finalización de nuestro programa.

En el siguiente ejemplo se produce un error sintáctico o parse.

```
<?php

function nombreApellidos(      // En esta línea dará error porque falta ')'
{
    return 'Javier Miras';
}
?>
```

En pantalla se mostrará el error de la siguiente forma.

```
Parse error: syntax error, unexpected '{', expecting variable (T_VARIABLE) in ...
```

Notice

Un `Notice`, o *aviso*, son los errores de menor nivel de gravedad. Aunque no paran el programa pueden hacer que no funcione como deseemos.

Veamos un ejemplo.

```
<?php
$nombre = 'Javier';

echo "<p>Me llamo $nobre</p>";
?>
```

El aviso que se produce es el siguiente.

```
Notice: Undefined variable: nobre in [...] on line 4

Me llamo
```

Como se puede ver el aviso no detiene la ejecución del programa, pero el resultado no es el esperado. En este caso está claro que hemos escrito mal la variable `$nombre`.

[44] http://php.net/manual/es/errorfunc.configuration.php

Los avisos, según donde se generen, pueden ser los siguientes.

- `E_NOTICE`: Se generan en tiempo de ejecución.
- `E_USER_NOTICE`: Lo generamos nosotros mediante la función `trigger_error()`[45].
- `E_STRICT`: Si se habilita, se emiten mensajes para alertar del uso de código que ha quedado obsoleto o que podría quedarlo en un futuro.
- `E_DEPRECATED`: Si se habilita, PHP generará avisos en tiempo de ejecución indicando que partes de código utilizado dejarán de funcionar en futuras versiones de PHP.

Warning

Una `Warning`, o *advertencia*, es un error que no repercute en la ejecución de nuestro programa. Es algo que estamos haciendo mal y que probablemente causará un error más adelante.

```
<?php
include('archivoInexistente.php');

$nombre = 'Javier';

echo "<p>Me llamo $nombre</p>";
?>
```

La advertencia que se produce es la siguiente.

```
Warning: include(archivoInexistente.php): failed to open stream: No such file or
directory in [...] on line 2

Warning: include(): Failed opening 'archivoInexistente.php' for inclusión
(include_path='.;\xampp\php\PEAR') in [...] on line 2

Me llamo Javier
```

Las advertencias, según donde se generen, pueden ser los siguientes.

- `E_WARNING`: Se generan en tiempo de ejecución.
- `E_CORE_WARNING`: Se generan en el núcleo de PHP durante el arranque.
- `E_COMPILE_WARNING`: Se generan en tiempo de compilación en el motor de Script Zend.
- `E_USER_WARNING`: Los generamos nosotros mediante la función `trigger_error()`.

Fatal error

Un `Fatal error`, o *excepción*,son errores que hacen que nuestro programa deje de funcionar. Son los que más nos tienen que preocupar y, por supuesto, los que más tenemos que controlar.

```
<?php
function nombreApellidos()
{
    return 'Javier Miras';
}

echo nombre_apellidos();
```

[45] http://php.net/manual/es/function.trigger-error.php

```
?>
```

El programa fallará porque la función `nombre_apellidos()` nos existe. El error que se mostrará es el siguiente.

```
Fatal error: Call to undefined function nombre_apellidos() in [...] on line 7
```

Los errores fatales, según donde se generen, pueden ser los siguientes.

- `E_ERROR`: Se generan en tiempo de ejecución.
- `E_CORE_ERROR`: Se generan en el núcleo de PHP durante el arranque.
- `E_COMPILE_ERROR`: Se generan en tiempo de compilación en el motor de Zend Engine.
- `E_USER_ERROR`: Los generamos nosotros mediante la función `trigger_error()`.
- `E_RECOVERABLE_ERROR`: Errores recuperables. Podemos gestionar el error mediante un gestor de errores, para ello utilizaremos la función set_error_handler()[46]. Si no está desarrollada la función, el error se convertirá en un `E_ERROR`.

[46] http://php.net/manual/es/function.set-error-handler.php

Gestionando los errores

Deberíamos mostrar todos los errores mientras estemos desarrollando, pero no cuando el programa esté en producción. En su lugar debemos gestionar los errores e informar al usuario que se ha producido y que acciones debemos tomar.

Imaginemos que queremos enviar un archivo por correo electrónico y la dirección proporcionada es incorrecta. Deberíamos manejar esta situación y pedir al usuario que nos proporcione una dirección correcta o cancelar el envío. Nunca debemos permitir que el programa deje de funcionar.

A continuación, vamos a ver dos técnicas para gestionar los errores. La primera es comprobar para evitar que se produzca el error y la segunda es gestionar el error una vez que se ha producido.

Comprobando errores

Existen ciertas condiciones que se nos puede dar en nuestro programa y que deberíamos chequearlas.

- Tipos de variables y valores: En este caso deberemos comprobar que el tipo de variable es el que esperamos, así como, el valor está dentro de un rango permitido. En un bucle `foreach` verificar que la variable es un vector o un objeto. No dividir por cero.
- Existencia de un recurso: Verificar la existencia de un fichero antes de trabajar con él. Comprobar que la variable no es NULL o no está asignada.
- Validación de datos suministrados: Comprobar que se llenaron todos los datos de un formulario. Verificar que no introducen código malicioso.

Chequeando los tipos y valores

Hemos visto que en PHP no se define el tipo de una variable, este cambia en función de la asignación que le hagamos. Cuando queramos chequear el tipo, antes de hacer una operación, utilizaremos la función `gettype()`[47] que nos indica qué tipo de variable es o alguna de las funciones específicas que nos proporciona PHP y que se indican a continuación.

Función	Descripción
is_numeric( )	Verdadero si número o cadena numérica
ctype_digit( )	Verdadero si todos los dígitos son caracteres numéricos
is_bool( )	Verdadero si la variable es un booleana
is_null( )	Verdadero si la variable es NULL
is_float( )	Verdadero si la variable es un número real
is_double( )	Es un alias de `is_float()`.
is_int( )	Verdadero si la variable es un número entero

[47] http://php.net/manual/es/function.gettype.php

is_string()	Verdadero si la variable es una cadena
is_object()	Verdadero si la variable es un objeto
id_array()	Verdadero si la variable es un array

Tabla 9. Funciones de chequeo de tipo y valor

Veamos un ejemplo.

```
<?php
function sumar($a, $b)
{
    if (is_numeric($a) && is_numeric($b))
        return $a + $b;

    trigger_error('Parámetros incorrectos.', E_USER_ERROR);
}

echo '<p>' .  sumar(3, 5) . '</p>';
echo '<p>' .  sumar(3, 'Hola') . '</p>';
?>
```

Existencia de recursos

Otro de los chequeos que debemos hacer es verificar la existencia de los recursos que utilizamos. PHP proporciona numerosas funciones para ello. Como siempre, cuando necesitemos verificar algo miramos primero si PHP nos proporciona una función.

Veamos un ejemplo.

```
<?php
$fichero = 'usuarios.php';

if (!file_exists($fichero))
    die("No existe el fichero $fichero.");

echo "Hola $nombre $apellidos.";
?>
```

En el ejemplo anterior, al no existir el fichero `usuarios.php` el programa finaliza mediante la función `die()`[48].

Otro ejemplo.

```
<?php
$fichero = NULL;

if (is_null($fichero))
    die('El valor de $fichero es nulo.');

if (!file_exists($fichero))
    die("El fichero $fichero no existe.");

echo "El fichero $fichero existe.";
?>
```

En esta ocasión la variable `$fichero` se la ha asignado el valor nulo, lo primero que hacemos en comprobar que está definida a nulo y después que el fichero existe.

[48] http://php.net/manual/es/function.die.php

Validación de datos suministrados

Cuando trabajamos con funciones y nos pasan datos mediante parámetros, siempre debemos comprobar que los datos suministrados son coherentes o están dentro de nuestro rango de trabajo.

Veamos un ejemplo.

```
<?php
function dividirEnteros($dividendo, $divisor, &$cociente, &$resto)
{
    if ($divisor == 0)
        return false;

    $cociente = intdiv($dividendo , $divisor);
    $resto    = $dividendo % $divisor;

    return true;
}

$dividendo = 7;
$divisor   = 0;
$cociente  = 0;
$resto     = 0;

if (!dividirEnteros($dividendo, $divisor, $cociente, $resto))
    die("Error al efectuar la división de $dividendo entre $divisor.");

echo "El resultado de dividir $dividendo entre $divisor es $cociente con un resto
      de $resto.";

?>
```

En este caso lo que debemos comprobar es que el divisor que pasamos a la función `dividirEnteros()` no es cero, pues en caso contrario nos daría un error. La función nos devolverá verdadero si ha podido hacer la división y falso en caso contrario. El error los gestionamos mediante una sentencia `if`.

Validar y sanear

Para *validar* (*validate*) y *sanear* (*sanitize*) datos, PHP nos proporciona la función `filter_var`[49].

Su sintaxis es la siguiente.

Sintaxis de filter_var
`filter_var(mixed $variable, int $filter = FILTER_DEFAULT, mixed $options = ?) : mixed;`

Los parámetros son los siguientes:

- Nombre de la variable: Variable a validar y sanear.
- Filtro: Filtro de validación o saneamiento que queremos aplicar.
- Opción: Opción a aplicar al filtro.

Para validar la entrada podemos utilizar los siguientes filtros:

- `FILTER_VALIDATE_BOOLEAN`: Valida la variable como un booleano.

[49] http://php.net/manual/es/function.filter-var.php

- `FILTER_VALIDATE_EMAIL`: Valida la variable como una dirección de correo electrónico correcta.
- `FILTER_VALIDATE_FLOAT`: Valida que la variable sea del tipo real.
- `FILTER_VALIDATE_INT`: Valida la variable como un número entero.
- `FILTER_VALIDATE_IP`: Valida la variable como una dirección IP.
- `FILTER_VALIDATE_REGEXP`: Valida la variable contra una expresión regular enviada en la variable de opciones.
- `FILTER_VALIDATE_URL`: Valida el valor coma una URL de acuerdo con la RFC 2396.

En cuanto a los filtros para sanear la entrada tenemos:

- `FILTER_SANITIZE_EMAIL`: Elimina todos los caracteres excepto las letras, los números y los siguientes caracteres !#$%&'*+-/=?^_`{|}~@.[].
- `FILTER_SANITIZE_ENCODED`: Codifica la cadena como una URL válida.
- `FILTER_SANITIZE_MAGIC_QUOTES`: Aplica la función `addslashes()`[50].
- `FILTER_SANITIZE_NUMBER_FLOAT`: Elimina todos los caracteres excepto los números, el signo más (`+`), el signo menos (`-`) y opcionalmente la coma (`,`), el punto (`.`) y el exponente (`e` ó `E`).
- `FILTER_SANITIZE_NUMBER_INT`: Elimina todos los caracteres excepto números, el signo más (`+`), el signo menos (`-`).
- `FILTER_SANITIZE_SPECIAL_CHARS`: Escapa caracteres HTML y caracteres con ASCII menor a 32.
- `FILTER_SANITIZE_STRING`: Elimina etiquetas, opcionalmente elimina o codifica caracteres especiales.
- `FILTER_SANITIZE_STRIPPED`: Alias del filtro anterior.
- `FILTER_SANITIZE_URL`: Elimina todos los caracteres excepto números, letras y los siguientes caracteres `$-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=`

Vamos a ver con uno ejemplo cómo validar números.

```
<?php

$var = 'dos';

echo ((filter_var($var, FILTER_VALIDATE_INT) === false) ?
      '<p>$var=' . $var . '  contiene un valor incorrecto.</p>' :
      '<p>$var=' . $var . '  contiene un valor correcto.</p>');

$var = 2;

echo ((filter_var($var, FILTER_VALIDATE_INT) === false) ?
      '<p>$var=' . $var . ' contiene un valor incorrecto.</p>' :
      '<p>$var=' . $var . ' contiene un valor correcto.</p>');

$var = '3,14159';
```

[50] http://php.net/manual/es/function.addslashes.php

```
echo ((filter_var($var, FILTER_VALIDATE_FLOAT) === false) ?
      '<p>$var=' . $var . '  contiene un valor incorrecto.</p>' :
      '<p>$var=' . $var . '  contiene un valor correcto.</p>');

$var     = '3,14159';
$options = array('options'=>array('decimal'=>','));

echo ((filter_var($var, FILTER_VALIDATE_FLOAT, $options) === false) ?
      '<p>$var=' . $var . ' contiene un valor incorrecto.</p>' :
      '<p>$var=' . $var . ' contiene un valor correcto.</p>');
?>
```

Por último, en el siguiente ejemplo se muestra cómo se sanea un número y un email.

```
<?php

$var = '2dos';

echo filter_var($var, FILTER_SANITIZE_NUMBER_INT);

$email = 'aj(miras)@ig//formacion.com';

echo filter_var($email, FILTER_SANITIZE_EMAIL);

?>
```

Para el saneamiento de cadenas tenemos las siguientes banderas (*flags*) que se pueden utilizar con el filtro `FILTER_SANITIZE_STRING`.

- `FILTER_FLAG_NO_ENCODE_QUOTES`: No codificará las comillas simples ni dobles.
- `FILTER_FLAG_STRIP_LOW`: Elimina caracteres cuyo varlor ASCII sea menor a 32.
- `FILTER_FLAG_STRIP_HIGH`: Elimina caracteres cuyo valor ASCII sea mayor a 127.
- `FILTER_FLAG_ENCODE_LOW`: Codifica caracteres cuyo valor ASCII sea menor a 32.
- `FILTER_FLAG_ENCODE_HIGH`: Codifica caracteres cuyo valor ASCII sea mayor a 127.
- `FILTER_FLAG_ENCODE_AMP`: Codifica ampersands (&).

Veamos un ejemplo de saneamiento de cadenas.

```
<?php
$text = '<p>"Ejemplo de dobles comillas ñ"</p>';

echo filter_var($text, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_HIGH);
?>
```

Excepciones

Una *excepción* (*exception*) es un evento que cambia el flujo normal de nuestro programa cuando se produce un error. El manejo de las excepciones puede hacer que nuestro código sea más fiable y robusto.

Lanzando una excepción - throw

Cuando nos encontremos con un error que hemos controlado lanzaremos nuestra excepción mediante la palabra reservada `throw`. Su sintaxis es la siguiente.

Sintaxis de throw

```
throw new Exception('Texto indicando el error producido.',
                    [Código error a mostrar],
                    [Excepción previa]);
```

Veamos un ejemplo.

```
<?php
function dividirEnteros($dividendo, $divisor, &$cociente, &$resto)
{
    if ($divisor == 0)
        throw new Exception('División por cero.');

    $cociente = intdiv($dividendo , $divisor);
    $resto    = $dividendo % $divisor;
}
?>
```

Al encontrarnos con un `$divisor` igual a cero lanzaremos una excepción, creando una nueva instancia de la clase `Exception()`[51], la cual parará el flujo normal del programa y producirá un error.

Algunos de los métodos más importantes de la clase `Exception` y que debemos conocer son los siguientes:

- getMessage(): Devuelve la cadena de error que se le pasa al constructor de la clase.
- getPrevious: Excepción previa.
- getCode(): Devuelve el entero que representa el código de error que se le pasa al constructor de la clase.
- getFile(): Devuelve el fichero donde se ha producido la excepción.
- getLine(): Devuelve la línea donde se ha producido la excepción.
- getTrace(): Devuelve un vector multidimensional con información del fichero y línea donde se ha producido la excepción, además de la función donde se ha producido la misma y sus argumentos.
- getTraceAsString(): Devuelve la información de la función anterior, pero en formato cadena.

Capturar una excepción - try/catch/finally

Para capturar las excepciones que se puedan producir utilizaremos la sentencia `try/catch/finally`. Su sintaxis es la siguiente.

```
Sintaxis de try/catch

try
{
    Líneas de código que pueden producir una excepción.
    ....
}
catch (Exception $e)
{
    Líneas de código con las acciones a realizar.
    ....
}
finally
{
```

[51] http://php.net/manual/es/class.exception.php

```
    Acciones de código que se realizarán siempre
}
```

Veamos un ejemplo.

```
<?php
function dividirEnteros($dividendo, $divisor, &$cociente, &$resto)
{
    if ($divisor == 0)
        throw new Exception('División por cero.');

    $cociente = intdiv($dividendo , $divisor);
    $resto    = $dividendo % $divisor;
}

$dividendo = 7;
$divisor   = 0;
$cociente  = 0;
$resto     = 0;

try
{
    dividirEnteros($dividendo, $divisor, $cociente, $resto);

    echo "El resultado de dividir $dividendo entre $divisor es $cociente
          con un resto de $resto.";
}
catch (Exception $e)
{
    echo 'Error al efectuar la división de $dividendo entre $divisor: ' .
         $e->getMessage();
}
?>
```

Al capturar la excepción no se producirá un error fatal, de esta forma, continuaremos con la ejecución de nuestro programa y tomaremos las acciones que consideremos oportunas o bien las que el usuario nos indique.

La opción del `finally` no es obligatoria, pero nos será de utilidad cuando trabajamos, por ejemplo, con archivos. En el `finally` se pondría el cierre del fichero.

```
<?php
// Abrimos el archive. Si hay algún error, abortamos el script.
$archivo = fopen("archivo.txt", "r") or die("No se puede abrir el archivo.");

try
{
    // Mostramos carácter a carácter hasta EOF
    while (!feof($archivo))
    {
        echo fgetc($archivo);
    }
}
catch (Exception $e)
{
    // Si se produce un error, lo mostramos en pantalla
    echo "Se ha producido un error: $e->getMessage()".
}
finally
{
    // Finalmente cerramos el archivo.
    fclose($archivo);
}
?>
```

Nuestra propia Exception

Podemos crearnos nuestra propia clase `Exception`, para ello la heredaremos y añadiremos funciones adicionales.

Veamos un ejemplo.

```
class UrlException extends Exception
{
   public function getMensaje()
   {
        // Mensaje de error
        $errorMsg = 'Error en la línea ' . $this->getLine()
                    . ', en el archivo ' . $this->getFile()
                    . ': <b>' . $this->getMessage()
                    . '</b> no es una URL válida.';

        return $errorMsg;
    }
}

$url = "URL incorrecta";

try
{
    if (!strpos($url, ".com"))
        throw new Exception("$url no es un dominio .com");

    if (filter_var($url, FILTER_VALIDATE_URL) === FALSE)
        throw new urlException($url);
}
catch (UrlException $e)
{
    echo $e->getMensaje();
}
catch (Exception $e)
{
    echo $e->getMessage();
}
```

En el ejemplo anterior se puede observar cómo capturar excepciones múltiples.

Ejercicios

81. Crea una que calcule el cuadrado de un área. Si el parámetro que se le pasa es negativo, deberá lanzar una excepción con el mensaje: "Debe introducir un número positivo". Prueba el programa con el siguiente vector (2, -5, 8)

82. Crea un programa que capture la excepción lanzada en el ejercicio anterior. Se mostrará el fichero, línea y mensaje de la siguiente forma: "Error en la línea X en el archivo Y: **Mensaje**."

Programación Orientada a Objetos

Uno de los objetivos de los programadores es crear código fácil de leer y mantener, que no contenga errores y escribirlo lo más rápido posible. La *Programación Orientada a Objetos* (*Object-oriented programming*) o POO, proporciona las herramientas necesarias para logarlo.

Además, la POO permite la implementación de técnicas avanzadas de programación con el *Modelo Vista Controlador* (*Model-View-Controler*) o MVC. Esta técnica de desarrollo software permite separar nuestra aplicación en tres capas: Interacción con la base de datos, presentación de la información y control del sistema.

Introducción a la POO

La POO permite organizar mejor los programas, aumenta la consistencia, reduce la redundancia y la complejidad, incrementando la flexibilidad y la seguridad. La POO debe guardar ciertas características que la identifican y diferencian de otros paradigmas de programación. Estas características se describen a continuación.

- **Abstracción.** Aislar un elemento de su contexto. Define las características esenciales de un objeto y determina su comportamiento.
- **Encapsulamiento**. Reunir en un mismo nivel de abstracción todos los elementos que puedan considerarse pertenecientes a una misma entidad.
- **Modularidad**. Permite dividir una aplicación en varias partes más pequeñas (llamados módulos), independientes unas de otras.
- **Ocultación**. Aislar los objetos del exterior, protegiendo sus propiedades para no ser modificadas por aquellos que no tengan derecho a acceder a las mismas.
- **Polimorfismo**. Capacidad de dar a diferentes objetos la posibilidad de contar con métodos, propiedades y atributos de igual nombre, pero con comportamientos.
- **Herencia**. Relación existente entre dos o más clases, donde una es la principal, o clase "madre", y otras dependen (heredan) de ella, o clase "hija".
- **Recolector de basura**. El entorno de objetos destruye automáticamente aquellos objetos que no se van a utilizar más.

Clases y objetos

La *clase* (*class*[52]) es el núcleo de la POO. La clase define como es un *objeto* (*object*), que información va a contener y que acciones puede realizar. Los objetos de una clase comparten un mismo estado (valores), identidad (propiedades) y comportamiento (métodos o mensajes).

La sintaxis de una clase es la siguiente.

```
Sintaxis de una clase
```

[52] http://php.net/manual/es/language.oop5.php

```
class NombreClase
{
    // Propiedades
    // Métodos
}
```

Las clases siguen las mismas reglas de nomenclatura que las variables, excepto que la primera letra la pondremos en mayúscula.

Cuando queramos trabajar con una clase tendremos que *instanciar* (*instantiate*) o crear un objeto. Su sintaxis es la siguiente.

Sintaxis de instanciación de una clase

```
$nombreObjeto = new NombreClase;
```

Para el ejemplo anterior tendremos.

```
<?php

class Usuario
{
}

$usuario = new Usuario;
?>
```

Herencia

La *herencia* o *extensión* (*extend*) es una característica que representa la relación entre dos clases, en la cual una clase hija hereda las características de una clase madre. Su sintaxis es la siguiente.

Sintaxis de herencia de una clase

```
class NombreClaseMadre
{
    // Propiedades clase madre
    // Métodos clase madre
}

class NombreClaseHija extends NombreClaseMadre
{
    // Propiedades de la clase hija, más las propiedades públicas y protegidas
       de la clase madre
    // Métodos de la clase hija, más los métodos públicos y protegidos de la
       clase madre
}
```

Clases abstractas

Las clases *abstractas* (*abstracts*) son aquellas que no necesitan instanciarse, son el esqueleto de una clase que será heredada y desarrollada posteriormente por la clase hija.

Sintaxis de una clase abstracta

```
abstract class NombreClaseAbstracta
{
    // Propiedades
    // Métodos
}
```

La finalidad de estas clases es indicar las características de la misma para que, posteriormente, las desarrollen las clases heredadas.

Clases finales

Una clase *final* es aquella que no puede ser heredada por otra clase. Su sintaxis es la siguiente.

Sintaxis de una clase final

```
final class NombreClaseFinal
{
    // Propiedades
    // Métodos
}
```

Propiedades o atributos

Las *propiedades* (*properties*[53]) o *atributos* (*attibute*) son variables que contiene los datos de un objeto. Su sintaxis es la siguiente.

Sintaxis de una propiedad

```
[public, protected, private] $nombrePropiedad;
```

La visibilidad de la propiedad indica la forma en la que podemos acceder a la misma:

- Pública (`public`): Podemos acceder desde cualquier parte de la aplicación sin restricciones de ningún tipo.
- Protegida (`protected`): Podemos acceder desde la misma clase o desde la clase que la hereda.
- Privada (`private`): Sólo podemos acceder desde la clase donde se ha definido.

Por defecto todas las propiedades son públicas, a menos que se indique lo contrario. Por buenas prácticas de programación es aconsejable declarar la visibilidad de todas las propiedades, aunque sean públicas.

La POO más estricta indica que todas las propiedades de una clase deben ser privadas, con el fin de garantizar la integridad, ocultación y encapsulación de los datos.

```
<?php
class Usuario
{
    // Propiedades
    public   $nombre          = 'Javier';
    rotected $apellidos       = 'Miras';
    private  $fechaNacimiento = '08/02/...';
}
?>
```

Propiedades estáticas

Las *propiedades estáticas* (*static properties*) son aquellas a las que podemos acceder sin necesidad de instanciar la clase y, como indica su nombre, su valor no lo podemos modificar. Su sintaxis es la siguiente.

[53] http://php.net/manual/es/language.oop5.properties.php

Sintaxis de una propiedad estática

```
[public, protected, private] static $nombrePropiedadEstática = ValorEstático;
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    public  $nombre          = 'Javier';
    public  $apellidos       = 'Miras';
    private $fechaNacimiento = '08/02/...';

    public static $ciudadNacimiento = 'Alicante';
}
?>
```

Acceso a las propiedades desde dentro de la clase

Para acceder a las propiedades desde dentro de una clase utilizaremos la siguiente sintaxis.

Sintaxis de acceso a una propiedad desde dentro de una clase

```
$this->nombrePropiedad;
```

La pseudo variable `$this` hace referencia al objeto donde está definida la propiedad.

Si la variable es estática utilizaremos la siguiente sintaxis.

Sintaxis de acceso a una propiedad estática desde dentro de una clase

```
self::$nombrePropiedadEstática;
parent::$nombrePropiedadEstática;
```

Para acceder a la propiedad estática utilizaremos el operador de resolución de ámbito (`::`) anteponiendo `self`, si se trata de una propiedad de la misma clase, o `parent`, si la propiedad es heredada.

Acceso a las propiedades desde fuera de la clase

Para acceder a las propiedades dentro de fuera de la clase utilizaremos la siguiente sintaxis.

Sintaxis de acceso a una propiedad desde fuera de una clase

```
$objeto->nombrePropiedad;
```

Si la variable es estática utilizaremos la siguiente sintaxis.

Sintaxis de acceso a una propiedad estática desde fuera de una clase

```
NombreClase::$nombrePropiedadEstática;
```

Para acceder a la propiedad estática utilizaremos el operador de resolución de ámbito (`::`) anteponiendo el nombre de la clase.

```
<?php
class Usuario
{
    public  $nombre          = 'Javier';
    public  $apellidos       = 'Miras';
    private $fechaNacimiento = '08/02/...';
```

```
    public static $ciudadNacimiento = 'Alicante';
}

echo '<p>' . Usuario::$ciudadNacimiento . '</p>';

$usuario = new Usuario;

echo '<p>' . $usuario->nombre . '</p>';
?>
```

Constantes

Las constantes de una clase se definen igual que el resto de las constates y sólo pueden tener la visibilidad pública.

Cuando queramos definir una constante y no queremos que accedan desde fuera de la clase o desde las clases heredadas, no nos queda más alternativa que declarar la constante como propiedad estática.

Métodos

Los *métodos* (*methods*) son las funciones de una clase. Los utilizaremos para dar operatividad a una clase o para acceder a las propiedades de esta. Su sintaxis es la siguiente.

Sintaxis de un método

```
[public, protected, private] function nombreMétodo(parámetro1,
                                                   parámetro2,
                                                   ...,
                                                   parámetroN=valor por defecto)
{
    Líneas de código;
    ....

    return valor;
}
```

La visibilidad de los métodos es exactamente la misma que para las propiedades.

Cuando queramos acceder a variables estáticas, tendremos que declarar nuestro método como estático, en caso contrario produciría un error.

```
<?php
class Usuario
{
    public  $nombre          = 'Javier';
    public  $apellidos       = 'Miras';
    private $fechaNacimiento = '08/02/...';

    public static $ciudadNacimiento = 'Alicante';

    // Métodos
    public function getFechaNacimiento() { return $this->fechaNacimiento; }
    public static function getCiudadNacimiento() { return self::$ciudadNacimiento; }
}

echo '<p>' . Usuario::$ciudadNacimiento . '</p>';
echo '<p>' . Usuario::getCiudadNacimiento() . '</p>';

$usuario = new Usuario;

echo '<p>' . $usuario->nombre . '</p>';
echo '<p>' . $usuario->getFechaNacimiento() . '</p>';
?>
```

En PHP existen los denominados *métodos mágicos* (*magic methods*[54]) que nos permiten ahorrarnos mucho tiempo de desarrollo de código y aportan mucha funcionalidad a las clases.

__construct

El método mágico `__construct()`[55], o método *constructor de la clase*, se invoca de manera automática al instanciar una clase. Lo utilizaremos para inicializar el objeto. Su sintaxis es la siguiente.

Sintaxis de un constructor

```
function __construct(parámetro1,
                     parámetro2,
                     ...,
                     parámetroN=valor por defecto)
{
    Líneas de código;
    ....
}
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    function __construct($nombre, $apellidos, $fechaNacimiento)
    {
        $this->nombre          = $nombre;
        $this->apellidos       = $apellidos;
        $this->fechaNacimiento = $fechaNacimiento;
    }
}

$usuario = new Usuario('Javier', 'Miras', '08/02....');
?>
```

En el ejemplo anterior a la hora de construir el objeto usuario le pasamos los parámetros de `$nombre`, `$apellidos` y `$fechaNacimiento`. Estos son recogidos con el constructor para inicializar las propiedades de la clase.

Los constructores de la clase madre no se invocan de forma automática, por lo que se tendremos que invocarlos implícitamente desde el constructor de la clase hija. Su sintaxis es la siguiente.

Sintaxis de invocación constructor clase madre

```
parent::__construct($paramentrosContructorClaseMadre);
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    function __construct($nombre, $apellidos, $fechaNacimiento)
    {
        $this->nombre          = $nombre;
```

[54] http://php.net/manual/es/language.oop5.magic.php
[55] http://php.net/manual/es/language.oop5.decon.php#object.construct

```
        $this->apellidos       = $apellidos;
        $this->fechaNacimiento = $fechaNacimiento;
    }
}

class Hija extends Usuario
{
    function __construct($nombre, $apellidos, $fechaNacimiento)
    {
        parent::  construct($nombre, $apellidos, $fechaNacimiento);
    }
}

$hija = new Hija('Javier', 'Miras', '08/02....');
?>
```

__destruct

El método `__destruct()`[56] es otro método mágico y es el *destructor de la clase*. Se invoca de manera automática al salir la clase del ámbito en el que fue declarada. Lo utilizaremos para destruir valores del objeto. Su sintaxis es la siguiente.

Sintaxis de un destructor

```
function __destruct()
{
    Líneas de código;
    ....
}
```

De forma análoga a los constructores de la clase madre, los destructores hay que llamarlos de forma implícita. Su sintaxis es la siguiente.

Sintaxis de invocación destructor clase madre

```
parent::__destruct();
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    function __construct($nombre, $apellidos, $fechaNacimiento)
    {
        $this->nombre          = $nombre;
        $this->apellidos       = $apellidos;
        $this->fechaNacimiento = $fechaNacimiento;
    }

    function __destruct()
    {
        echo 'Acciones a tomar en el destructor.';
    }
}

class Hija extends Usuario
{
    function __construct($nombre, $apellidos, $fechaNacimiento)
    {
        parent::__construct($nombre, $apellidos, $fechaNacimiento);
    }

    function __destruct()
```

[56] http://php.net/manual/es/language.oop5.decon.php#object.destruct

```
    {
        parent::__destruct();
    }
}

$hija = new Hija('Javier', 'Miras', '08/02....');
?>
```

Setters y __set

Los métodos de *asignación* (*setter*) son métodos desarrollados por nosotros para asignar el valor a las propiedades cuando estas han sido declaradas como privadas. Su sintaxis es la siguiente.

Sintaxis de un setter

```
public function setNombrePropiedad($valor)
{
    // Acciones a realizar antes de asignar el valor
    // como comprobar que todo es correcto.

    $this->nombrePropiedad = $valor;
}
```

Por convenio, a los métodos setter se le antepone la palabra `set` para indicar la acción de asignación.

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    public function setNombre($val)          { $this->nombre          = $val; }
    public function setApellidos($val)       { $this->apellidos       = $val; }
    public function setFechaNacimiento($val) { $this->fechaNacimiento = $val; }
}

$usuario = new Usuario;

$usuario->setNombre('Javier');
$usuario->setApellidos('Miras');
$usuario->setFechaNacimiento('08/02/....');
?>
```

PHP proporciona el método mágico __set()[57] que nos permite asignar un valor a cualquier propiedad de una clase. Su sintaxis es la siguiente.

Sintaxis del método mágico __set

```
public function __set($propiedad, $valor)
{
    // Acciones a realizar antes de asignar el valor
    // como comprobar que todo es correcto.

    $this->$propiedad = $valor;
}
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
```

[57] http://php.net/manual/es/language.oop5.overloading.php#object.set

```
    private $apellidos;
    private $fechaNacimiento;

    public function   set($propiedad, $val)
    {
        if (property_exists(__CLASS__, $propiedad))
            $this->$propiedad = $val;
    }
}

$usuario = new Usuario;

$usuario->__set('nombre', 'Javier');
$usuario->__set('apellidos', 'Miras');
$usuario->__set('fechaNacimiento', '08/02/....');

$usuario->nombre          = 'José';
$usuario->apellidos       = 'García';
$usuario->fechaNacimiento = '14/10/....';
?>
```

Se puede observar que al desarrollar `__set()` podemos acceder a una variable privada como si fuese pública, pero tomando el control sobre la acción de asignación.

La función `property_exists()`[58] no indica si la propiedad existe en la clase. `__CLASS__`[59] es una constante predefinida que indica el nombre de la clase.

Getters y __get

Los métodos de *obtención* (*getter*) son métodos desarrollados por nosotros para obtener el valor de las propiedades cuando estas han sido declaradas como privadas. Ya los hemos visto en ejemplo anteriores. Su sintaxis es la siguiente.

Sintaxis de un getter

```
public function getNombrePropiedad()
{
    // Acciones a realizar antes de devolver el valor.

    return $this->nombrePropiedad;
}
```

Por convenio, a los métodos getter se le antepone la palabra `get` para indicar la acción de obtención.

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    public function setNombre($val)          { $this->nombre          = $val; }
    public function setApellidos($val)       { $this->apellidos       = $val; }
    public function setFechaNacimiento($val) { $this->fechaNacimiento = $val; }

    public function getNombre()          { return $this->nombre; }
    public function getApellidos()       { return $this->apellidos; }
    public function getFechaNacimiento() { return $this->fechaNacimiento; }
}

$usuario = new Usuario;

$usuario->setNombre('Javier');
```

[58] http://php.net/manual/es/function.property-exists.php
[59] http://php.net/manual/es/language.constants.predefined.php

```
$usuario->setApellidos('Miras');
$usuario->setFechaNacimiento('08/02/....');

echo '<p>' . $usuario->getNombre() . '</p>';
echo '<p>' . $usuario->getApellidos() . '</p>';
echo '<p>' . $usuario->getFechaNacimiento() . '</p>';
?>
```

Al igual que con `__set()`, PHP proporciona el método mágico __get()[60] que nos permite acceder a cualquier propiedad de una clase. Su sintaxis es la siguiente.

Sintaxis del método mágico __get

```
public function __get($propiedad)
{
    // Acciones a realizar antes de devolver el valor.

    return $this->$propiedad;
}
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    public function __set($propiedad, $val)
    {
        if (property_exists(__CLASS__, $propiedad))
            $this->$propiedad = $val;
    }

    public function __get($propiedad)
    {
        if (property_exists(__CLASS__, $propiedad))
           return $this->$propiedad;

        return NULL;
   }
}

$usuario = new Usuario;

$usuario->__set('nombre', 'Javier');
$usuario->__set('apellidos', 'Miras');
$usuario->__set('fechaNacimiento', '08/02/....');

echo '<p>' . $usuario->__get('nombre') . '</p>';
echo '<p>' . $usuario->__get('apellidos') . '</p>';
echo '<p>' . $usuario->__get('fechaNacimiento') . '</p>';

$usuario->nombre          = 'José';
$usuario->apellidos       = 'García';
$usuario->fechaNacimiento = '14/10/....';

echo '<p>' . $usuario->nombre . '</p>';
echo '<p>' . $usuario->apellidos . '</p>';
echo '<p>' . $usuario->fechaNacimiento . '</p>';
?>
```

__clone

El método `__clone()`[61] es otro método mágico y se utiliza para clonar un objeto. Su sintaxis es la siguiente.

[60] http://php.net/manual/es/language.oop5.overloading.php#object.get
[61] https://www.php.net/manual/es/language.oop5.cloning.php

Sintaxis del método clone

```
function __clone()
{
    Líneas de código;
    ....
}
```

Veamos un ejemplo.

```
<?php
class Usuario
{
    private $nombre;
    private $apellidos;
    private $fechaNacimiento;

    function __construct($nombre, $apellidos, $fechaNacimiento)
    {
        $this->nombre          = $nombre;
        $this->apellidos       = $apellidos;
        $this->fechaNacimiento = $fechaNacimiento;
    }

    function __destruct()
    {
        echo 'Acciones a tomar en el destructor.';
    }

    function __clone()
    {
        $this->fechaNacimiento = "Ninguna";
    }
}

$usuario = new Usuario('Javier', 'Miras', '08/02....');

echo '<p>' . $usuario->getNombre() . '</p>';
echo '<p>' . $usuario->getApellidos() . '</p>';
echo '<p>' . $usuario->getFechaNacimiento() . '</p>';

$usuarioClonado = clone($usuario);

echo '<p>' . $usuarioClonado->getNombre() . '</p>';
echo '<p>' . $usuarioClonado->getApellidos() . '</p>';
echo '<p>' . $usuarioClonado->getFechaNacimiento() . '</p>';
?>
```

Ejercicios

83. Crea una clase llamada Persona, en el fichero "persona.php". Esta clase deberá tener un atributo "nombre". También deberá tener un método "SetNombre", con un parámetro que permita cambiar el valor del nombre. Finalmente, también tendrá un método "Saludar",que escribirá en pantalla "Hola, soy " seguido del nombre.

84. Para guardar información sobre libros, vamos a comenzar por crear una clase "Libro", que contendrá atributos "autor", "titulo", "ubicacion" y métodos Get y Set adecuados para leer su valor y cambiarlo. Crea una aplicación para probarla.

85. Crea una variante ampliada del ejercicio 82. En ella, la clase Persona no cambia. Se creará una nueva clase PersonaInglesa, en el fichero "personainglesa.php". Esta clase deberá heredar las características de la clase "Persona", y añadir un método "TomarTe", que escribirá en pantalla "Estoy tomando té". Para probarlo crea dos objetos de tipo Persona y uno de tipo PersonaInglesa, les asignará un nombre, les pedirá que saluden y pedirá a la persona inglesa que tome té.

86. Amplía el proyecto del ejercicio 83 crea una clase "Documento", de la que "Libro" heredará todos sus atributos y métodos. Ahora la clase Libro contendrá sólo un atributo "paginas" y con sus correspondientes Get y Set.

87. Amplia el proyecto del ejercicio 84. Crea una clase llamada "PersonaItaliana" y guárdala en el fichero "personaitaliana.php". El método "Saludar" mostrará en pantalla "Ciao".

88. Retoca el proyecto del ejercicio 85 para que los atributos de la clase "Documento" y de la clase "Libro" sean protegidos.

89. Ampliar las clases del ejercicio 85, para que todas ellas contengan constructores. Los constructores de casi todas las clases estarán vacíos, excepto el de "PersonaInglesa", que prefijará su nombre a "John". Crea también un constructor alternativo para esta clase que permita escoger cualquier otro nombre.

90. Amplia la clase "Libro" para que cuando se clone en el atributo "paginas" ponga "Sin definir".

Bases de Datos

Conectando con la base de datos

En este capítulo, aprenderemos los interfaces que comunican PHP con MySQL. Originalmente los métodos se llamaban con `mysql`[62]. Todavía podemos encontrar código con estos métodos, pero es aconsejable utilizar la nueva extensión `mysqli`[63], que nos proporciona más seguridad.

PHP Data Object (PDO[64]) es una extensión para PHP que conecta, no solo con MySQL, si no con otras bases de datos como PostgreSQL, SQLite y Oracle.

La forma tradicional de conectarnos con MySQL era mediante `mysql`, pero con `mysqli` tenemos las siguientes ventajas:

- Interfaz orientado a objetos: MySQLi tiene las siguientes clases `mysqli` para conectarnos con la base de datos, `sysql_stmt` para las consultas y `mysqli_result` para los resultados.
- Consultas preparadas: Cuando utilizamos una consulta más de una vez, es importante que sea una consulta preparada, de esta forma será mucho más rápida y segura frente a inyecciones.
- Múltiples consultas: Podemos procesar múltiples consultas en lugar de una a una.
- Transacciones: Podemos hacer transacciones a nuestra base de datos.

Conectando con MySQLi

Para conectar con MySQL necesitamos conocer el nombre del servidor, el nombre del usuario, su contraseña y la base de datos. Crearemos una instancia de la clase `mysqli` para establecer la misma.

Veamos un ejemplo.

```
<?php
$con = new mysqli('localhost', 'root', '12345', 'basedatos');

if ($con->connect_error)
    die('Error: ' . $con->connect_error);

echo 'Conectado a MySQL';
?>
```

Si se produce un error podemos acceder a él mediante la propiedad `connect_error`[65]. El único problema es que PHP también mostrará un error indicando el fichero y la fila. Para callar esto y que sólo se muestre nuestro error, antepondremos el símbolo de la arroba (`@`) delante de la instanciación de la clase `mysqli`.

```
<?php
$con = @new mysqli('localhost', 'root', '12345', 'basedatos');

if ($con->connect_error)
    die('Error: ' . $con->connect_error);
```

[62] http://php.net/manual/es/book.mysql.php
[63] http://php.net/manual/es/book.mysqli.php
[64] http://php.net/manual/es/book.pdo.php
[65] http://php.net/manual/es/mysqli.connect-error.php

```
echo 'Conectado a MySQL';
?>
```

Hemos visto cómo conectarnos con el estilo POO de `mysqli`. Hay también un tipo procedimental de `mysqli`. El siguiente ejemplo muestra como conectarse usando el estilo procedimental.

```
<?php
$con = @mysqli_connect('localhost', 'root', '12345', 'basedatos');

if (!$con)
    die('Error nº: ' . mysqli_connect_errno() . '. Error: ' . mysqli_connect_error());

echo 'Conectado a MySQL';
?>
```

Independientemente del estilo utilizado, POO o procedimental, utilizamos la variable `$con` que contiene la conexión a la base de datos. Otra forma de realizar la conexión es creando una clase que contenga la misma.

```
<?php
class BaseDatos
{
    private static $usuario     = 'root';
    private static $contrasenya = '12345';
    private static $baseDatos   = 'basedatos';
    private static $servidor    = 'localhost';

    protected static $con = NULL

    public static function getConexion()
    {
        self::$con = @new mysqli(self::$servidor,
                                 self::$usuario,
                                 self::$contrasenya,
                                 self::$baseDatos);

        if (self::$con->connect_error)
            die('Error: ' . self::$con->connect_error);

        return self::$con;
    }
}
?>
```

Observar que en lugar de utilizar `$this->` utilizamos `self::` puesto que estamos accediendo a una propiedad estática.

El único problema que se nos plantea es que cada vez que llamamos a nuestra clase creamos una conexión. Para evitar esto limitaremos nuestra clase a la creación de una *única instancia* (*singleton*).

```
<?php
class BaseDatos
{
    private static $usuario     = 'root';
    private static $contrasenya = '12345';
    private static $baseDatos   = 'basedatos';
    private static $servidor    = 'localhost';

    protected static $con = NULL;

    private function __construct()
    {
        // No hacemos nada.
    }

    private function __clone()
    {
        // No hacemos nada.
    }
```

```
    public static function getConexion()
    {
        if (!self::$con)
        {
            self::$con = @new mysqli(self::$servidor,
                                     self::$usuario,
                                     self::$contrasenya,
                                     self::$baseDatos);

            if (self::$con->connect error)
                die('Error: ' . self::$con->connect_error);
        }

        return self::$con;
    }
}
?>
```

El ejemplo anterior está con `mysqli` con clases, se puede hacer lo mismo con `mysqli` con procedimiento y con PDO.

Para evitar que nuestra clase se utilice mediante la creación de una instancia haremos que el constructor de la clase sea privado. Lo mismo hacemos con el método mágico `__clone()`, para evitar que nos clonen la clase. Fuera de la clase utilizaremos el método `getConexion()` mediante la resolución de ámbito de la clase.

Veamos un ejemplo.

```
<?php
$con = BaseDatos::getConexion();

// Resto de programa con acciones a realizar con la conexión.

$con->close();
?>
```

Conectando con PDO

La conexión `mysqli` está creada específicamente para trabajar con base de datos MySQL. El problema que nos podemos encontrar es que la base de datos con la que tengamos que trabajar no sea MySQL; que sea, por ejemplo, Oracle, PostgreSQL, Firebird, SQLite o Informix, entre otros. La solución es utilizar PDO[66], que nos permite hacer nuestras aplicaciones más portables hacia otros gestores de bases de datos.

A continuación, se muestra una lista de características de PDO.

- Orientado a objetos: PDO tiene la principal clase `PDO`, la clase `PDOStament` para las consultas, y la clase `PDOException` para los errores.
- Consultas preparadas: Cuando utilizamos una consulta más de una vez, es importante que sea una consulta preparada, de esta forma será mucho más rápida y segura frente a inyecciones.
- Transacciones: Podemos hacer transacciones a nuestra base de datos.

Trabajar con PDO es similar a trabajar con `mysqli`. Veamos un ejemplo.

```
<?php
try
{
    $con = new PDO('mysql:host=localhost;dbname=basedatos','root','12345');
```

[66] http://php.net/manual/es/book.pdo.php

```
    echo 'Conectado a MySQL';
}
catch(PDOException $e)
{
    die('Error: ' . $e->getMessage());
}

$con = NULL;
?>
```

Ejecutando sentencias MySQL

Una de las acciones que más vamos a realizar en nuestros programas PHP son las consultas y acciones de inserción, modificación y borrado de datos en las tablas MySQL. Para realizar estas operaciones siempre seguiremos los siguientes pasos:

- Crear una conexión a la base de datos.
- Crear una sentencia SQL.
- Ejecutar la sentencia SQL.
- Cerrar la conexión.

Al igual que cualquier programa de gestión de la base de datos MySQL, podemos utilizar la conexión para obtener una lista de las tablas de la base de datos. La clase `mysqli`[67] proporciona el método `query()`[68] al cual le pasaremos la sentencia a ejecutar, en este caso `SHOW TABLES`, y nos devuelve un objeto de la clase `mysqli_result`[69] con el resultado de la consulta.

```
<?php
$con = BaseDatos::getConexion();

if ($query = $con->query('SHOW TABLES'))
{
    $filas = $query->num_rows;

    echo "Tablas ($filas):<br />";

    while ($fila = $query->fetch_array())
    {
        echo $fila[0]. '<br />';
    }
}

$con->close();
?>
```

La propiedad `num_rows`[70] contiene el número de registros de nuestra consulta. Con el método `fetch_array()`[71] obtenemos un vector con los campos de cada registro. En este caso el primer elemento del vector contiene el nombre de la tabla. Este método encuentra solo la primera fila de nuestra consulta, para conseguir con el resto de las filas usamos un bucle `while`.

La sentencia podemos crearla en tiempo de ejecución mediante la creación de una cadena de caracteres y asignársela a una variable. En ocasiones la cadena puede ser extremadamente larga y puede llevarnos a cometer errores en su creación.

[67] http://php.net/manual/es/mysqli.summary.php
[68] http://php.net/manual/es/mysqli.query.php
[69] http://php.net/manual/es/class.mysqli-result.php
[70] http://php.net/manual/es/mysqli-result.num-rows.php
[71] http://php.net/manual/es/mysqli-result.fetch-array.php

Veamos un ejemplo.

```
$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, telefono,
        direccion, cp, poblacion, provincia FROM usuarios WHERE provincia
        IN ("Alicante", "Castellón", "Valencia")';
```

En ocasiones la cadena puede ser extremadamente larga y puede llevarnos a cometer errores en su creación. Lo más coherente es que la cadena se parte en varias subcadenas para su mejor comprensión.

```
$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
       . 'telefono, direccion, cp, poblacion, provincia '
       . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';
```

Consultas de selección

Las consultas de selección son aquellas que nos devuelven un conjunto de resultados como consecuencia de la ejecución de una sentencia `SELECT`.

Seleccionando con MySQLi

Para recorrer el resultado devuelto por una sentencia de selección mediante POO podemos utilizar los siguientes métodos:

- `fetch_assoc()`: Devuelve el resultado de una fila en un vector asociativo.
- `fetch_object()`: Devuelve el resultado de una fila en un objeto.
- `fetch_array()`: Devuelve el resultado de una fila en un vector.
- `fecth_all()`: Devuelve el resultado en un vector de dos dimensiones con todas las filas. La primera dimensión indica cada uno de los registros de la tabla y la segunda dimensión los campos.

Si la consulta es procedimental, tenemos las siguientes funciones:

- `mysqli_fetch_assoc()`: Devuelve el resultado de una fila en un vector asociativo.
- `mysqli_fetch_object()`: Devuelve el resultado de una fila en un objeto.
- `mysqli_fetch_array()`: Devuelve el resultado de una fila en un vector.
- `mysqli_fecth_all()`: Devuelve el resultado en un vector de dos dimensiones con todas las filas. La primera dimensión indica cada uno de los registros de la tabla y la segunda dimensión los campos.

Cuando obtenemos una fila de nuestra consulta mediante un vector (`fetch_array()` o `mysqli_fetch_array()`) o bien mediante una tabla entera (`fetch_all()` o `mysqli_fetch_all()`), podemos obtener los resultados de tres formas distintas:

- `MYSQLI_ASSOC`: Devuelve un vector indexado por los nombres de las columnas.
- `MYSQLI_NUM`: Devuelve un vector indexado por el número de columna, comenzando por la columna 0.

- `MYSQLI_BOTH` (Valor por defecto): Devuelve un vector indexado, tanto por nombre de columna como numéricamente comenzando por la columna 0.

Veamos un ejemplo utilizando POO.

```
<?php
$con = BaseDatos::getConexion();

$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
       . 'telefono, direccion, cp, poblacion, provincia '
       . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';

if ($query = $con->query($sql))
{
    while ($fila = $query->fetch_array(MYSQLI_BOTH))
        print_r($fila);
}
else
    echo "Error: " . $con->error;

$con->close();
?>
```

De forma procedimental escribiríamos nuestro código de la siguiente forma.

```
<?php
$con = @mysqli_connect('localhost', 'root', '12345', 'basedatos');

$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
       . 'telefono, direccion, cp, poblacion, provincia '
       . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';

if ($query = mysqli_query($con, $sql))
{
    while ($fila = mysqli_fetch_array($query, MYSQLI_NUM))
        print_r($fila);
}
else
    echo "Error: " . $mysqli_error;

mysqli close($con);
?>
```

Veamos ahora un ejemplo de cómo obtener toda la tabla completa, sin necesidad de recorrerla fila a fila, y asignarla a un vector mediante POO.

```
<?php
$con = BaseDatos::getConexion();

$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
       . 'telefono, direccion, cp, poblacion, provincia '
       . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';

if ($query = $con->query($sql))
{
    $tabla = $query->fetch_all(MYSQLI_ASSOC);

    if ($query->num rows > 0)
        print_r($tabla);
    else
        echo 'Nada que mostrar.';
}
else
    echo "Error: " . $con->error;

$con->close();
?>
```

De forma procedimental sería como se muestra en el siguiente ejemplo.

```
<?php
$con = @mysqli_connect('localhost', 'root', '12345', 'basedatos');
```

```
$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
       . 'telefono, direccion, cp, poblacion, provincia '
       . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';

if ($query = mysqli_query($con, $sql))
{
    $tabla = mysqli_fetch_all($query, MYSQLI_NUM);

    if (mysqli num rows($query) > 0)
        print_r($tabla);
    else
        echo 'Nada que mostrar.';
}
else
    echo "Error: " . $mysqli error;

mysqli_close($con);
?>
```

Seleccionando con PDO

En el caso de PDO, utilizaremos los métodos:

- `fetch()`: Devuelve el resultado de una fila.
- `fetchAll()`: Devuelve el resultado en un vector de dos dimensiones con todas la filas. La primera dimensión indica cada uno de los registros de la tabla y la segunda dimensión los campos.

Cuando obtenemos una fila de nuestra consulta mediante un vector (`fetch()`) o bien mediante una tabla entera (`fetchAll()`), podemos obtener los resultados de tres formas distintas:

- `PDO::FETCH_ASSOC`: Devuelve un vector indexado por los nombres de las columnas.
- `PDO::FETCH_NUM`: Devuelve un vector indexado por el número de columna, comenzando por la columna 0.
- `PDO::FETCH_BOTH` (Valor por defecto): Devuelve un vector indexado, tanto por nombre de columna como numéricamente comenzando por la columna 0.

Veamos un ejemplo utilizando PDO.

```
<?php
$con = new PDO('mysql:host=localhost;dbname=basedatos','root','12345');

$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
       . 'telefono, direccion, cp, poblacion, provincia '
       . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';

if ($query = $con->query($sql))
{
    while ($fila = $query->fetch(PDO::FETCH_ASSOC))
        print_r($fila);
}

$con = NULL;
?>
```

Este ejemplo nos muestra como acceder a los datos de una consulta fila a fila.

Veamos un ejemplo de cómo obtener una tabla completa y asignarla a un vector.

```
<?php
$con = new PDO('mysql:host=localhost;dbname=basedatos','root','12345');

$sql = 'SELECT usuario_id, nombre, apellidos, usuario, contrasenya, '
```

```
        . 'telefono, direccion, cp, poblacion, provincia '
        . 'FROM usuarios WHERE provincia IN ("Alicante", "Castellón", "Valencia")';

if ($query = $con->query($sql))
{
   $tabla = $query->fetchAll(PDO::FETCH_ASSOC);

   if ($query->rowCount() > 0)
       print_r($tabla);
   else
       echo 'Nada que mostrar.';
}
$con = NULL;
?>
```

Consultas de inserción

Vamos a ver la forma de realizar inserciones en nuestra base de datos

Insertando con MySQLi

Empecemos por una inserción utilizando POO.

```
<?php
$con = BaseDatos::getConexion();

$sql = 'INSERT INTO usuarios '
       . 'VALUES (NULL, "José", "García", "jose", "12345", "YYYYYYYYY", '
       . '"C/. Su calle nº 23", "03001", "Alicante", "Alicante")';

if ($con->query($sql) === true)
    echo 'Se ha insertado ' . $con->affected_rows . ' fila(s).<br/>' .
         'El id de la última inserción es: ' . $con->insert_id . '.';
else
    echo 'Error (' . $con->error . '): ' . $con->error;

$con->close();
?>
```

Veamos cómo se hace utilizando la forma procedimental.

```
<?php
$con = @mysqli_connect('localhost', 'root', '12345', 'basedatos');

$sql = 'INSERT INTO usuarios '
       . 'VALUES (NULL, "José", "García", "jose", "12345", "YYYYYYYYY", '
       . '"C/. Su calle nº 23", "03001", "Alicante", "Alicante")';

if (mysqli_query($con, $sql) === true)
    echo 'Se ha insertado ' . mysqli_affected_rows($con) . ' fila(s).<br/>' .
         'El id de la última inserción es: ' . mysqli_insert_id($con) . '.';
else
    echo 'Error (' . mysqli_errno($con) . '): ' . mysqli_error($con);

mysqli_close($con);
?>
```

Insertando con PDO

Para insertar en una tabla utilizando PDO se muestra en el siguiente ejemplo.

```
<?php
$con = new PDO('mysql:host=localhost;dbname=basedatos','root','12345');

$sql = 'INSERT INTO usuarios '
       . 'VALUES (NULL, "José", "García", "jose","12345", "YYYYYYYYY", '
       . '"C/. Su calle nº 23", "03001", "Alicante", "Alicante")';

$stm = $con->prepare($sql);
```

```
if (!$stm->execute())
    print_r($stm->errorInfo());
else
    echo 'Se ha insertado ' . $stm->rowCount() . ' fila(s).<br/>' .
         'El id de la última inserción es: ' . $con->lastInsertId() . '.';

$con = NULL;
?>
```

Consultas de actualización

Vamos a ver la forma de realizar actualizaciones en nuestra base de datos.

Actualizando con MySQLi

Empecemos por una actualización utilizando POO.

```
<?php
$con = BaseDatos::getConexion();

$sql = 'UPDATE usuarios SET telefono="XXXXXXXXXX" WHERE usuario_id=10';

if ($con->query($sql) === true)
    echo 'Se ha(n) modificado ' . $con->affected_rows. ' fila(s).';
else
    echo 'Error (' . $con->errno . '): ' . $con->error;

$con->close();
?>
```

Veamos cómo se hace utilizando la forma procedimental.

```
<?php
$con = @mysqli_connect('localhost', 'root', '12345', 'basedatos');

$sql = 'UPDATE usuarios SET telefono="XXXXXXXXXX" WHERE usuario_id=10';

if (mysqli_query($con, $sql) === true)
    echo 'Se ha(n) modificado ' . mysqli_affected_rows($con) . ' fila(s).';
else
    echo 'Error (' . mysqli_errno($con) . '): ' . mysqli_error($con);

mysqli close($con);
?>
```

Actualizando con PDO

La actualización de una fila o conjunto de filas de una tabla utilizando PDO se muestra en el siguiente ejemplo.

```
<?php
$con = new PDO('mysql:host=localhost;dbname=basedatos','root','12345');

$sql = 'UPDATE usuarios SET telefono="XXXXXXXXXX" WHERE usuario_id=10';

$stm = $con->prepare($sql);

if (!$stm->execute())
    print_r($stm->errorInfo());
else
    echo 'Se ha(n) modificado ' . $stm->rowCount() . ' fila(s).';

$con = NULL;
?>
```

Consultas de borrado

Vamos a ver la forma de realizar borrados en nuestra base de datos.

Borrando con MySQLi

Empecemos por un borrado utilizando POO.

```
<?php
$con = BaseDatos::getConexion();

$sql = 'DELETE FROM usuarios WHERE usuario_id=10';

if ($con->query($sql) === true)
    echo 'Se ha(n) borrado ' . $con->affected_rows. ' fila(s).';
else
    echo 'Error (' . $con->errno . '): ' . $con->error;

$con->close();
?>
```

Veamos cómo se hace utilizando la forma procedimental.

```
<?php
$con = @mysqli_connect('localhost', 'root', '12345', 'basedatos');

$sql = 'DELETE FROM usuarios WHERE usuario_id=10';

if (mysqli_query($con, $sql) === true)
    echo 'Se ha(n) borrado ' . mysqli_affected_rows($con) . ' fila(s).';
else
    echo 'Error (' . mysqli_errno($con) . '): ' . mysqli_error($con);

mysqli_close($con);
?>
```

Borrando con PDO

El procedimiento de borrado en una tabla utilizando PDO se muestra en el siguiente ejemplo.

```
<?php
$con = new PDO('mysql:host=localhost;dbname=basedatos','root','12345');

$sql = 'DELETE FROM usuarios WHERE usuario_id=10';

$stm = $con->prepare($sql);

if (!$stm->execute())
    print_r($stm->errorInfo());
else
    echo 'Se ha(n) borrado ' . $stm->rowCount() . ' fila(s).';

$con = NULL;
?>
```

Ejercicios

91. Crea una base de datos con el nombre "ejercicios". Los datos de acceso a la base de datos estarán en fichero llamado "constantes.php". Crea la siguiente tabla "usuarios":

Campo	Tipo
usuario_id	int(11), PK, autoincremental
nombre	varchar(32)
apellidos	varchar(100)

usuario	varchar(16)
contrasenya	char(64)

92. Crea una clase llamada "Conexión", guárdala como "conexión.php" que cree una conexión a la base de datos creada en el ejercicio 90. La clase devolverá la conexión. Si no se puede conectar deberá mostrar el correspondiente error "Error (*Nº de error*): *Mensaje con el error producido*.". La conexión se realizará con MySQLi con clases.

93. Crea una clase llamada "Usuarios" y guárdala como "usuarios.php". Esta clase realizará las siguientes operaciones, utilizando MySQLi con clases, sobre la tabla "usuarios" de la base de datos "ejercicios":

    - Consulta: Se podrá consultar todos los usuarios o un usuario en concreto. Los datos se mostrarán en una tabla en HTML.
    - Inserción: Se insertará un usuario.
    - Modificación: Se modificará el usuario indicado.
    - Borrado: Se borrará el usuario indicado.

94. Realizar los ejercicios 91 y 92 utilizando PDO.

95. ¿Qué son las sentencias parametrizadas o preparadas? Indica sus ventajas. Pon un ejemplo utilizando la base de datos "ejercicios".

Ficheros

El manejo de ficheros es bastante simple en PHP. Vamos a ver de forma básico el manejo de ficheros de texto.

fopen

La primera función es `fopen` que nos abre un fichero. Su sintaxis es:

Sintaxis de fopen

```
fopen($filename, $mode): resource
```

El primer parámetro, filename, es el nombre del fichero a abrir. El segundo parámetro, `mode`, es el modo de apertura.

La función devuelve el puntero al fichero abierto o falso si se produce un error.

Los modos de apertura son los siguientes:

- r Modo de solo lectura. Se abre el fichero y el cursor se coloca al principio de este, permitiendo leerlo hasta el final.
- r+ Modo de lectura/escritura. Se abre el fichero y el cursor se coloca al principio de este, permitiendo leer o escribir en el fichero.
- w Modo de solo escritura. Se crea el fichero si no existiese, y, si existe, se borra todo su contenido, se sitúa el cursor al principio del fichero permitiendo escribir.
- w+ Modo de escritura/lectura. Si el fichero no existe, se crea, y si existiese, se borra su contenido, se sitúa el cursor al principio del fichero permitiendo leer y escribir.
- a Modo de añadido. Abre el fichero, sitúa el cursor al final de este y permite escribir. Si el fichero no existe, lo crea, pero si existe, no borra su contenido.
- a+ Modo de añadido/lectura. Sitúa el cursor al final del fichero y permite leer y escribir. Si el fichero no existe, lo crea, pero si existe, no borra su contenido.

fclose

La función que nos cierra un fichero es `fclose`. Su sintaxis es:

Sintaxis de fclose

```
fclose($filename): boole
```

La función cierra el fichero, `filename`, abierto previamente. Su la función cierra el fichero devuelve `true`, en caso conrario `false`.

fputs

La función `fputs` escribe una línea de texto en un fichero. Su sintaxis es:

Sintaxis de fgets

```
fputs($filename, $string): int
```

La función escribe la cadena de texto `string` en el fichero `filename`. Devuelve el número de bytes escrito o false si se ha producido un error.

fgets

La función `fgets` lee una línea de texto de un fichero. Su sintaxis es:

Sintaxis de fgets

```
fgets($filename): string
```

La función lee una cadena de texto del fichero `filename`, en caso de error devuelve `false`.

Ejemplo completo

El siguiente es un ejemplo, sencillo, pero ilustrativo del funcionamiento de los ficheros de texto en PHP.

```
<?php

try
{
    // Abrimos el fichero en modo escritura.
    $fichero = fopen("fichero.txt","w");

    // Escribimos la primera línea de texto.
    $string = "1ª línea de texto\n";

    fputs($fichero, $string);

    // Escribimos la segunda línea de texto.
    $string = "2ª línea de texto\n";

    fputs($fichero, $string);

    // Escribimos la tercera línea de texto.
    $string = "3ª línea de texto\n\n";

    fputs($fichero, $string);

    // Cerramos el fichero.
    fclose($fichero);

    // Volvemos a abrir el fichero, esta vez en modo añadir.
    $fichero = fopen("fichero.txt","a");

    // Añadimos la cuarta línea de texto directamente.
    fputs($fichero, "4ª línea de texto, añadida con modo \"a\"\n");

    // Añadimos la quinta línea de texto directamente.
    fputs($fichero, "5ª línea de texto, añadida con modo \"a\"");

    // Cerramos el fichero...
    fclose($fichero);
```

```
    // y lo volvemos a abrir en modo lectura.
    $fichero = fopen("fichero.txt", "r");

    // Leemos línea a línea, y la mostramos, hasta que devuelva false.
    while ($cadena = fgets($fichero))
    {
        echo $cadena . "<br/>";
    }
}
catch (exception $e)
{
    // Si hay algún problema mostramos mensaje de error.
    echo $e->getMessage();
}
finally
{
    // Cerramos el fichero.
    fclose($fichero);
}

?>
```

Descarga de ficheros

El siguiente ejemplo muestra cómo descargar un archivo desde nuestro servidor.

```
<?php
$rutaFichero = 'files/archivo.txt';
$fichero     = basename($rutaFichero);

header("Content-Type: application/octet-stream");
header("Content-Length: " . filesize($rutaFichero));
header("Content-Disposition: attachment; filename=$fichero");

readfile($rutaFichero);
?>
```

Ejercicios

96. Crea un programa que lea de un fichero la siguiente secuencia escrita en CSV. Utiliza la función `explode`[72].

```
Javier;Miras;ajmiras@igformacion.com;DAW
José;García;jgarcia@igformacion.com;DAW
Juan;Latorre;jlatorre@igformacion.com;DAW
```

97. Crea un programa que lea de un fichero la secuencia del ejercicio anterior. Utiliza la función `fgetcsv`[73].

98. Crea un programa que modifique la secuencia del ejercicio anterior, cambiando DAW por DAM y guardándolas en un fichero. Utiliza la función `implode`[74].

99. Crea un programa que modifique la secuencia del ejercicio anterior, cambiando DAW por DAM y guardándolas en un fichero. Utiliza la función `fputcsv`[75].

100. Crea un programa que descargue un fichero PDF. El fichero estará en la carpeta PDFs del dominio de nuestro programa. En el programa especifica que el archivo es PDF.

[72] https://www.php.net/manual/es/function.explode.php
[73] https://www.php.net/manual/es/function.fgetcsv.php
[74] https://www.php.net/manual/es/function.implode.php
[75] https://www.php.net/manual/es/function.fputcsv.php

Formularios HTML y PHP

En el capítulo anterior hemos visto cómo realizar consultas, inserciones, actualizaciones y borrados de nuestra base de datos. Lo usual es que sea el cliente quien nos proporcione la información para llevar a cabo esas acciones.

Esta información nos la facilitará a través de un formulario escrito en HTML, redirigiendo la acción a nuestro programa PHP, bien por un método `POST` o bien por un método `GET`.

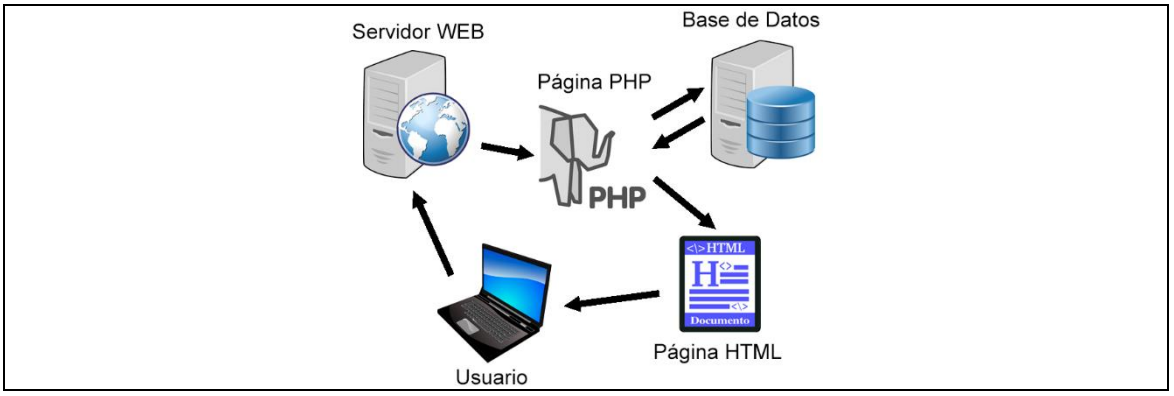

Figura 1. Formularios HTML y PHP

POST

Mediante el método `POST` tenemos las siguientes características de envío.

- Todos los nombres de los campos y valores viajan en el cuerpo de la petición HTTP.
- No hay límite de longitud de los datos enviados.
- Los datos no están expuestos en la URL o logs del servidor.
- Las operaciones no se pueden cachear.
- Permite en envió de información de tipo binario. Lo que nos permite enviar ficheros.
- La codificación por defecto es `application/x-www-form-urlencoded`, y `multipart/form-data` para envió de ficheros.

GET

Para le método `GET` tendremos las siguientes características.

- Todos los nombres de los campos y valores se envían en la URL.
- Al mandarse por la URL el navegador los almacena en su historial. Esto hace que la información sea sensible.
- La longitud máxima de la URL es de 2048 caracteres.

- Las operaciones `GET` pueden ser cacheadas. Por lo que, varias llamadas a un servicio pueden resultar una única petición.
- Los datos enviados pueden quedar almacenados en los ficheros log del servidor
- Solo se pueden enviar datos ASCII, no se puede enviar datos binarios como ficheros.
- La única codificación posible es `application/x-www-form-urlencoded`.

Procesando la información de un formulario HTML

El típico formulario escrito en HTML se muestra a continuación.

```
<form action="contacto.php?accion=1" method="post">
    <p>
        <label for="nombre">Nombre</label>
        <input type="text" name="nombre" id="nombre" />
    </p>
    <p>
        <label for="apellidos">Apellidos</label>
        <input type="text" name="apellidos" id="apellidos" />
    </p>
    <p>
        <label for="usuario">Usuario</label>
        <input type="text" name="usuario" id="usuario" />
    </p>
    <p>
        <label for="contrasenya">Contraseña</label>
        <input type="text" name="contrasenay" id="contrasenya" />
    </p>
    <p>
        <label for="direccion">Dirección</label>
        <input type="text" name="direccion" id="direccion" />
    </p>
    <p>
        <label for="cp">C.P.</label>
        <input type="text" name="cp" id="cp" />
    </p>
    <p>
        <label for="poblacion">Población</label>
        <input type="text" name="poblacion" id="poblacion" />
    </p>
    <p>
        <label for="provincia">Provincia</label>
        <input type="text" name="provincia" id="provincia" />
    </p>
    <p>
        <label for="telefono">Teléfono</label>
        <input type="text" name="telefono" id="telefono" />
    </p>
    <p>
        <label for="email">Email</label>
        <input type="text" name="email" id="email" />
    </p>
    <p>
        <label for="fechanacimiento">Fecha nacimiento</label>
        <input type="text" name=" fechanacimiento " id=" fechanacimiento " />
    </p>
    <p>
        <input type="submit" id="enviar" value="Enviar"/>
    </p>
</form>
```

Como podemos observar estamos utilizando los dos métodos de paso de información a nuestro programa. Por `POST` le pasaremos toda la información del formulario, mientras que por `GET` le indicamos la acción a realizar.

Para obtener los valores que nos envía podemos utilizar las variables super globales `$_GET`[76] y `$_POST`[77].

Veamos un ejemplo.

```
<?php
echo '<p>Los valores pasados por GET son los siguientes:</p>';
print_r($_GET);

echo '<p>Y por POST nos han pasado: </p>';
print_r($_POST);
?>
```

Validando y saneando la información de un formulario HTML

Aunque hoy en día muchas de las validaciones se realizan en el lado del cliente, nunca debemos dar por buenos los datos que nos envían. Por lo tanto, siempre tendremos que *validar* (*validate*) y *sanear* (*sanitize*) todo lo que nos envían, de esta forma prevenimos ataques a nuestro sitio web, como *Inyecciones SQL* (*SQL Inyection*) o la *Encriptación Cruzada de Sitios* (*Cross Site Scripting – XSS*)

Para validar y sanear los datos que nos envían desde el formulario utilizaremos la función `filter_input`[78], a la función le podemos pasar los mismos parámetros que le pasamos a la función `filter_var`.

Veamos, a continuación, como procesamos la información que nos han facilitado desde un formulario HTML.

```
<?php
$accion = filter_input(INPUT_GET, 'accion', FILTER_VALIDATE_INT);

if (($accion === null) || ($accion === false))
{
    echo 'Acción incorrecta: ' . $_GET["accion"];
    die();
}

$nombre = filter_input(INPUT_POST, 'nombre',
                       FILTER_SANITIZE_STRING, FILTER_FLAG_NO_ENCODE_QUOTES);

// Resto de validación y saneamiento...

$email = filter_input(INPUT_POST, 'email', FILTER_SANITIZE_EMAIL);

$email = filter_var($email, FILTER_VALIDATE_EMAIL);

if (($email === null) || ($email === false))
{
    echo 'Email incorrecto: ' . $_POST["email"];
    die();
}
?>
```

Subir ficheros al servidor

Para subir fichero en PHP comenzaremos creando un formulario que nos permita seleccionar el fichero a subir, en este caso una imagen.

```
<form action="subirimagen.php" method="post" enctype="multipart/form-data">
```

[76] http://php.net/manual/es/reserved.variables.get.php
[77] http://php.net/manual/es/reserved.variables.post.php
[78] http://php.net/manual/es/function.filter-input.php

```
    Seleccione una imagen a subir:
    <input type="file" name="imagen" id="fichero">
    <input type="submit" value="Imagen a subir" name="submit">
</form>
```

Hay que tener varias consideraciones en cuenta para poder subir un archivo al servidor:

- El método de envío debe ser `post`.
- Hay que poner el atributo `enctype="multipart/form-data"`.
- Hay que poner un `input` de tipo `file` para el usuario pueda seleccionar el fichero a subir.

El formulario llamará al script `subirimagen.php` que es el que se encargará de gestionar la subida. En el siguiente ejemplo se muestra la subida de una imagen.

```
<?php
$carpetaDestino = 'imagenes/';
$ficheroDestino = $carpetaDestino . basename($_FILES["imagen"]["name"]);
$correcto       = true;
$tipoImagen     = strtolower(pathinfo($ficheroDestino, PATHINFO_EXTENSION));
$nombreImagen   = htmlspecialchars(basename($_FILES["imagen"]["name"]));

// Comprobamos que el fichero es una imagen.
if (isset($_POST["submit"]))
{
    $esImagen = getimagesize($_FILES["imagen"]["tmp_name"]);

    if ($esImagen !== false)
    {
      echo "La imagen $nombreImagen es de tipo - {$esImagen["mime"]}.";
    }
    else
    {
      echo "El fichero $nombreImagen no es una imagen.";
      $correcto = false;
    }
}

// Verificamos si la imagen ya se había subido.
if (file_exists($ficheroDestino))
{
    echo "La imagen $nombreImagen existe, no se subirá.";
    $correcto = false;
}

// Chequeamos el tamaño máximo permitido.
if ($_FILES["imagen"]["size"] > 500000)
{
    echo "La imagen $nombreImagen excede el tamaño máximo permitido (500000).";
    $correcto = false;
}

// Sólo permitimos estos formatos.
if ($tipoImagen != 'jpg' && $tipoImagen != 'jpeg' && $tipoImagen != 'png')
{
    echo 'Sólo se permiten imágenes JPG/JPEG y PNG.';
    $correcto = false;
}

// Verificamos que no ha habido error.
if ($correcto == true)
{
    // Movemos la imagen de la carpeta temporal a la carpeta definitiva
    if (move_uploaded_file($_FILES["imagen"]["tmp_name"], $ficheroDestino))
        echo "La imagen $nombreImagen se ha subido.";
    else
        echo "Ha habido un error al guardar la imagen $nombreImagen .";
}
?>
```

Ejercicios

101. Modifica el formulario HTML del ejemplo para que se validen los datos en el formulario. Guárdalo en "contacto.html". El nombre no debe contener más de 50 caracteres. Los apellidos como máximo 100 caracteres. Los caracteres de la contraseña no deben visualizarse. El teléfono sólo números. El código postal sólo números y como máximo 5. El email debe ser correcto. La fecha de nacimiento deberá mostrar un calendario para introducirla.

102. Realiza la validación de los datos del formulario HTML "contacto.html" en el archivo "contacto.php".

103. Amplía el formulario "contacto.html" para que suba una foto del usuario.

104. Amplía "contacto.php" para que guarde la foto del usuario.

105. Crear una aplicación para subir ficheros de Microsoft Word. El tamaño máximo del fichero será de 30 Mb.

Cookies

Una cookie es un pequeño archivo que se guarda en el equipo del usuario el cual contendrá información. La cookie se envía desde el navegador del usuario al servidor cada vez que este visite una página web, lo cual nos permitirá recuperar los datos almacenados.

Las cookies tienen una gran desventaja porque pueden ser rechazadas por el usuario, su número por dominio está limitado por los navegadores y su tamaño también es limitado. Nunca se debe almacenar información confidencial o sensible mediante cookies.

setcookie

La función `setcookie`[79] nos permite crear una cookie en el equipo del usuario.

Sintaxis de setcookie

```
setcookie(
    string $name,
    string $value = "",
    int $expires = 0,
    string $path = "",
    string $domain = "",
    bool $secure = false,
    bool $httponly = false
): bool

setcookie(string $name, string $value = "", array $options = []): bool
```

La función crearé la cookie llamada `$name` con el valor `$value`. Si todo es correcto devolverá `TRUE` en caso contrario devolverá `FALSE`.

Podemos proporcionar los valores de todos los parámetros, o bien utilizar el vector `$options`, con los siguientes valores:

- `$expires`: Fecha de caducidad de la cookie (timestamp Unix). Si no se indica nada la cookie expirará al finalizar la sesión.
- `$path`: Ruta de acceso en el servidor donde está disponible la cookie. Si se utiliza /, la cookie estará disponible en todo el dominio. Si se configura como */fotos/*, sólo estará disponible para el directorio */fotos/* y sus subdirectorios.
- `$domain`: Dominio, o subdomino, al que se reenvía la cookie. Por ejemplo: *.midominio.com* (con un punto al principio) permite que la cookie esté disponible para todos los subdominios de *midominio.com*.
- `$secure`: Indica que la cookie sólo se enviará en una conexión segura HTTPS.
- `$httponly`: Si es TRUE la cookie sólo se puede transmitir por HTTP, por lo que no será accesible por otros lenguajes como JavaScript.
- `$samesite`: Debe ser uno de los siguientes valores: None, Lax o Strict[80].

[79] https://www.php.net/manual/es/function.setcookie.php
[80] https://developer.mozilla.org/es/docs/Web/HTTP/Cookies#cookies_samesite_experimental_inline

Si omitimos un valor en el vector `$options`, este tomará el mismo que los valores por defecto de los parámetros explícitos.

Veamos un ejemplo.

```
<?php
$opciones = array (
    'expires' => time() + 60*60*24*30, // Expira en 30 días
    'path' => '/',                     // Todo el dominio
    'domain' => '.midominio.com',      // Dominio y subdominios
    'secure' => true,                  // Sólo HTTPS
    'httponly' => true,                // No permitimos otros lenguajes
    'samesite' => 'Strict'             // Sólo accessible desde midominio.com
    );

setcookie('MiCookie', 'Valor de la cookie', $opciones);
?>
```

Para acceder al contenido de una cookie guardada en el ordenador del usuario utilizaremos la variable superglobal `$_COOKIE`.

```
<?php

if (isset($_COOKIE["MiCookie"]))
    echo $_COOKIE["MiCookie"];

?>
```

Ejercicios

106. Crea un programa que verifique si existe la cookie "IGFormacion". En el caso de que no exista que dé el correspondiente mensaje de error.

107. Crea una cookie con las siguientes características:

    - Expira en 5 minutos
    - Para todo el dominio
    - No es necesaria la conexión segura HTTPS

108. Crea un programa que verifique la existencia de la cookie anterior y muestre en pantalla los datos de esta.

Sesiones

El ámbito de una variable es el script en el que está definida y se destruye con este. Si queremos mantener dicho valor entre diferentes scripts podemos utilizar las Sesiones (Sessions).

Para trabajar con los valores de la sesión utilizaremos la variable superglobal `$_SESSION`.

PHP ofrece una serie de funciones para gestionarlas:

Función	Cometido
`session_start`	Abre una nueva sesión o reactiva la sesión actual.
`session_id`	Devuelve, o modifica, el identificador de la sesión.
`session_destroy`	Elimina la sesión.
`session_name`	Devuelve, o modifica, el nombre de la variable utilizada para almacenar el identificador de la sesión.
`session_status`	Devuelve el estado actual de una sesión.
`session_abort`	Anula las modificaciones efectuadas en los datos de sesión y termina la sesión.
`session_reset`	Reinicializa los datos de sesión a sus valores iniciales.

session_start

La función `session_start`[81] es la primera función que se debe poner en nuestro script para trabajar con sesiones.

Sintaxis de session_start

```
session_start([array opciones]): bool
```

La función devuelve `TRUE` si se ha podido crear la función y `FALSE` en caso contrario.

Si se proporciona el parámetro opciones, sobrescribirá las directivas de configuración de sesiones[82] establecidas actualmente. Las claves no deben incluir el prefijo session.

El siguiente script abre una sesión y guarda un texto en la variable `$_SESSION["cadena"]`.

```
<?php
session_start();

$_SESSION["cadena"] = 'Hola a todos';
?>
```

En otro script se abre una sesión y recogemos el texto asignado a dicha variable.

```
<?php
session_start();

echo $_SESSION["cadena"];
?>
```

[81] https://www.php.net/manual/es/function.session-start.php
[82] https://www.php.net/manual/es/session.configuration.php

session_id

La función `session_id`[83] la utilizaremos para obtener o establecer el id de la sesión actual.

Sintaxis de session_id
`session_id([string id=?]): string`

La función devuelve el id asociado a la sesión actual o la cadena vacía ("") si no existe la sesión.

Si se proporciona el parámetro id, reemplazará el id de la sesión actual.

```
<?php
session_start();

echo session_id();
?>
```

session_destroy

La función `session_destroy`[84] la utilizaremos para destruir la información de la sesión actual.

Sintaxis de session_destroy
`session_destroy(): bool`

La función devuelve `TRUE` si se ha destruido la sesión y `FALSE` en caso contrario.

```
<?php
session_start();

$_SESSION["cadena"] = 'Hola a todos';

echo $ SESSION["cadena"];

session_destroy();
?>
```

La función `session_destroy` no libera la variable `$_SESSION["cadena"]`, debemos hacerlo utilizando la función `unset($_SESSION["cadena"])`.

Nunca debemos hacer `unset($_SESSION)` ya que esto deshabilitará el registro de variables de sesión a través del array superglobal `$_SESSION`.

session_name

La función `session_name`[85] la utilizaremos para obtener o modificar el nombre de la sesión actual.

Sintaxis de session_name
`session_name(string $name=?): string`

[83] https://www.php.net/manual/es/function.session-id.php
[84] https://www.php.net/manual/es/function.session-destroy.php
[85] https://www.php.net/manual/es/function.session-name.php

La función devuelve el nombre de la sesión actual. Si proporcionamos el nombre de la sesión `name`, este se actualizará y nos devolverá el nombre antiguo.

```
<?php
session_start();

echo session_name('nuevo_nombre');

session_destroy();
?>
```

session_status

La función `session_status`[86] devuelve el estado actual de la sesión.

Sintaxis de session_status

```
session_status(): int
```

Los valores que devuelve son:

- PHP_SESSION_DISABLED: Las sesiones están deshabilitadas.
- PHP_SESSION_NONE: Las sesiones están habilitadas, pero no existe ninguna.
- PHP_SESSION_ACTIVE: Las sesiones están habilitadas, y existe una.

```
<?php
session_start();

echo session_status();

session_destroy();
?>
```

session_abort

La función `session_abort`[87] finaliza la sesión y deshace los cambios del vector de sesión.

Sintaxis de session_abort

```
session_abort(): bool
```

La función devuelve `TRUE` si se ha destruido la sesión y `FALSE` en caso contrario.

```
<?php
echo session_save_path('Nuestra ruta');

session_start();

echo session_save_path();

session_abort();

echo session_save_path();
?>
```

[86] https://www.php.net/manual/es/function.session-status.php
[87] https://www.php.net/manual/es/function.session-abort.php

session_reset

La función `session_reset`[88] reinicializa los datos de la sesión a sus valores iniciales, a diferencia de `session_abort`, esta función anula las modificaciones inmediatamente en el script actual.

Sintaxis de session_reset

```
session_reset(): void
```

Veamos un ejemplo:

```
// pagina1.php

<?php
session_start();

$_SESSION["nombre"] = 'Javier Miras';

echo $_SESSION["nombre"];

echo '<br/>';

echo '<a href="pagina2.php">Ir a página 2</a>';
?>
```

```
// pagina2.php

<?php
session_start();

$_SESSION["nombre"] = 'Javier Miras';

echo $_SESSION["nombre"];

$_SESSION["nombre"] = 'No soy Javier Miras';

session_reset();

echo $_SESSION["nombre"];
?>
```

Ejercicios

109. Crea un programa llamado "sesion1.php" donde se cree una variable sesión llama "error". Inicialízala a la cadena vacía.

110. Crea un programa llamado "session2.php" donde verificará que existe la variable sesión "error", si existe le dará el valor "Error: Existe un error.".

111. Crea un programa llamado "sesion3.php" donde verificará si existe la variable sesión "error". En el caso de que exista la mostrará en pantalla. Si no existe mostrará: "No existe la variable sesión Error.".

112. Crea un programa llamado "sesion4.php" que destruya, si existe, la variable sesión "error".

113. Crea un programa que le pida al usuario un nombre y una contraseña. Si el nombre es "Javier" y la contraseña "12345" deberá ir a una página que diga "Logueado". Si no es correcto deberá volver a pedir el usuario y la contraseña. Dispone de 3 intentos. Si al tercer intento no lo ha puesto correctamente, deberá ir a una página que diga "Ponte en contacto

[88] https://www.php.net/manual/es/function.session-reset.php

con el servicio técnico.". Pensad bien el nombre de cada archivo antes de comenzar la práctica y analizad cómo podemos guardarnos los intentos.

Enviar correo electrónico

PHP ofrece una manera simple y directa enviar mensajes y archivos a través del correo electrónico, sin necesidad de instalar librerías adicionales. Todo lo realizaremos con la función `mail`[89].

mail

La función `mail` envía mensaje de correo electrónico. Su sintaxis es la siguiente.

Sintaxis de mail

```
mail(string $to, string $subject, string $message,
     string $additional_headers = ?,
     string $additional_parameters = ?): bool
```

La función devuelve `TRUE` si se ha enviado con éxito el mensaje y `FALSE` en caso contrario. Que la función haya tenido éxito no significa que el mensaje haya alcanzado el destino indicado.

Los parámetros de la función son los siguientes:

- `$to`: Destinatario(s) del mensaje.
- `$subject`: Título del mensaje.
- `$message`: Mensaje a enviar. Cada línea debería separarse con un CRLF (`"\r\n"`) y no ocupar más de 70 caracteres.
- `$additional_headers`: Se utiliza para añadir una cabecera extra, por ejemplo: *From*, *CC* y *BCC*, siempre separado por CRLF (`"\r\n"`).
- `$additional_parameters`: Puede usarse para indicar opciones adicionales como opciones de línea de comandos al programa que está configurado para usarse cuando se envía el mensaje.

Veamos un ejemplo.

```
<?php
// Destinatarios. En este caso dos destinatarios, observar la coma (,) entre ambos.
$para = 'estudiantesDAW@igformacion.com,estudiantesDAM@igformacion.com';

// Asunto del mensaje
$asunto = 'Recordatorio de entrega de actividad.';

// mensaje
$mensaje =
    'Recordatorio de entrega de actividad' . "\r\n" .
    'El próximo lunes vence el plazo para la entrega de la actividad 1.' . "\r\n" .
    'Un saludo,' . "\r\n" .
    'Javier Miras' . "\r\n";

// Repetimos la dirección de los estudiantes para mandarla en formato "limpio"
// y añadimos uno nuevo.
$cabecera .= 'To: Estudiantes DAW <estudiantesDAW@igformacion.com>,' .
             'Estudiantes DAM <estudiantesDAM@igformacion.com>,' .
             'FP IG <formacionprofesional@igforamcion.com>' . "\r\n";
```

[89] https://www.php.net/manual/es/function.mail.php

```
// Remitente del mensaje.
$cabecera .= 'From: Javier Miras <ajmiras@igformacion.com>' . "\r\n";

// Direcciones que reciben copia del mensaje.
$cabecera .= 'Cc: administracion@igformacion.com' . "\r\n";

// Direcciones ocultas que reciben copia del mensaje.
$cabecera .= 'Bcc: jefeestudios@igformacion.com' . "\r\n";

// Prioridad del mensaje. Desde 1, prioridad más alta, hasta 4, prioridad más baja.
$cabecera .= 'X-Priority:1' . "\r\n";

// Enviamos el mensaje.
if (mail($para, $asunto, $mensaje, $cabecera) == true)
    echo 'Mensaje enviado correctamente.';
else
    echo 'Error al enviar el mensaje.';
?>
```

MIME

El formato `MIME` nos permitirá enviar un mensaje en formato HTML, adjuntar ficheros, etc. Este formato está compuesto de tres líneas:

- `MIME-Version`: El mensaje es te tipo MIME y especifica la versión.
- `Content-Type`: Tipo de contenido.
- `Content-Transfer-Encoding`: Tipo de cifrado.

Los tipos de contenido habituales son los siguientes:

- `text/plain`: Mensaje en texto plano. Se puede especificar un juego de caracteres en `charset`, por ejemplo, `utf-8`.
- `text/html`: Mensaje en formato HTML. Se puede especificar un juego de caracteres en `charset`, por ejemplo, `utf-8`.
- `image/jpg`: Imagen en formato jpg.
- `application/octet-stream`: Datos binarios.

Los cifrados más habituales son los siguientes:

- `7bit`: Cifrado en 7 bits.
- `8bit`: Cifrado en 8 bits, utilizado para mantener los acentos y caracteres especiales.
- `base64`: Cifrado para datos binarios: imágenes, ficheros, etc.

Veamos un ejemplo.

```
<?php
// Destinatarios.
$para = 'estudiantesDAW@igformacion.com,estudiantesDAM@igformacion.com';

// Asunto del mensaje
$asunto = 'Nueva actividad.';

// Mensaje HTML
$contenidoHTML = '
<html>
<body>
  <h1>Nueva actividad</h1>
  <p>En el fichero adjunto está la nueva actividad</p>
  <p>Un saludo,</p>
  <p>Javier Miras</p>
</body>
</html>
';
```

```
// Repetimos la dirección de los estudiantes para mandarla en formato "limpio"
// y añadimos uno nuevo.
$cabecera = 'To: Estudiantes DAW <estudiantesDAW@igformacion.com>,' .
            'Estudiantes DAM <estudiantesDAM@igformacion.com>,' .
            'FP IG <formacionprofesional@igforamcion.com>' . "\r\n";

// Remitente del mensaje.
$cabecera .= 'From: Javier Miras <ajmiras@igformacion.com>' . "\r\n";

// Direcciones que reciben copia del mensaje.
$cabecera .= 'Cc: administracion@igformacion.com' . "\r\n";

// Direcciones ocultas que reciben copia del mensaje.
$cabecera .= 'Bcc: jefeestudios@igformacion.com' . "\r\n";

// Prioridad del mensaje. Desde 1, prioridad más alta, hasta 4, prioridad más baja.
$cabecera .= 'X-Priority:1' . "\r\n";

// Límite que indica cada una de las partes que componen el mensaje.
$limiteMIME = '==Limite_Multiparte_x'. md5(time()) . 'x';

// Encabezados para enviar un mensaje en formato HTML y con un fichero adjunto.
$cabecera .= 'MIME-Version: 1.0' . "\r\n";
$cabecera .= 'Content-Type: multipart/mixed; boundary="' . $limiteMIME . '"' . "\r\n";

// Parte del mensaje HTML
$mensaje  = '--' . $limiteMIME . "\r\n";
$mensaje .= 'Content-Type: text/html; charset="UTF-8"' . "\r\n";
$mensaje .= 'Content-Transfer-Encoding: 8bit' . "\r\n\n";
$mensaje .= $htmlContent . "\r\n";

// Fichero a adjuntar.
$fichero = 'actividad.pdf';
$fp      = @fopen($fichero,"rb");
$datos   = @fread($fp, filesize($fichero));
@fclose($fp);
$datos   = chunk_split(base64_encode($datos));

// Parte del mensaje donde se adjunta el fichero
$mensaje .= '--' . $limiteMIME. "\r\n";
$mensaje .= 'Content-Type: application/octet-stream;' .
            ' name=' . basename($fichero) . "\r\n";
$mensaje .= 'Content-Description: ' . basename($fichero) . "\r\n";
$mensaje .= 'Content-Disposition: attachment;' . "\r\n";
$mensaje .= 'filename="' . basename($file) . '";' .
            ' size=' . filesize($file). ';' . "\r\n";
$mensaje .= 'Content-Transfer-Encoding: base64;' . "\r\n";
$mensaje .= $datos . "\r\n";

// Fin del mensaje
$mensaje .= '--' . $limiteMIME .'--';

// Enviamos el mensaje.
if (mail($para, $asunto, $mensaje, $cabecera) == true)
    echo 'Mensaje enviado correctamente.';
else
    echo 'Error al enviar el mensaje.';
?>
```

Ejercicios

114. Amplia el ejemplo anterior para mandar dos fichero adjuntos.

Modelo-Vista-Controlador

En este punto vamos a desarrollar un sistema Modelo-Vista-Controlador (MVC) básico para entender el funcionamiento de este patrón de diseño. La utilización del MVC nos facilitará la portabilidad de las aplicaciones y un mantenimiento más organizado de las mismas.

El MVC separa la programación entres aspectos fundamentales:

- El Modelo: Es la lógica de negocio de la aplicación y se encarga de la gestión de los datos.
- La Vista: Es la presentación de los datos al usuario.
- El Controlador: Es el encargado de gestionar los eventos y las comunicaciones entre el Modelo y la Vista.

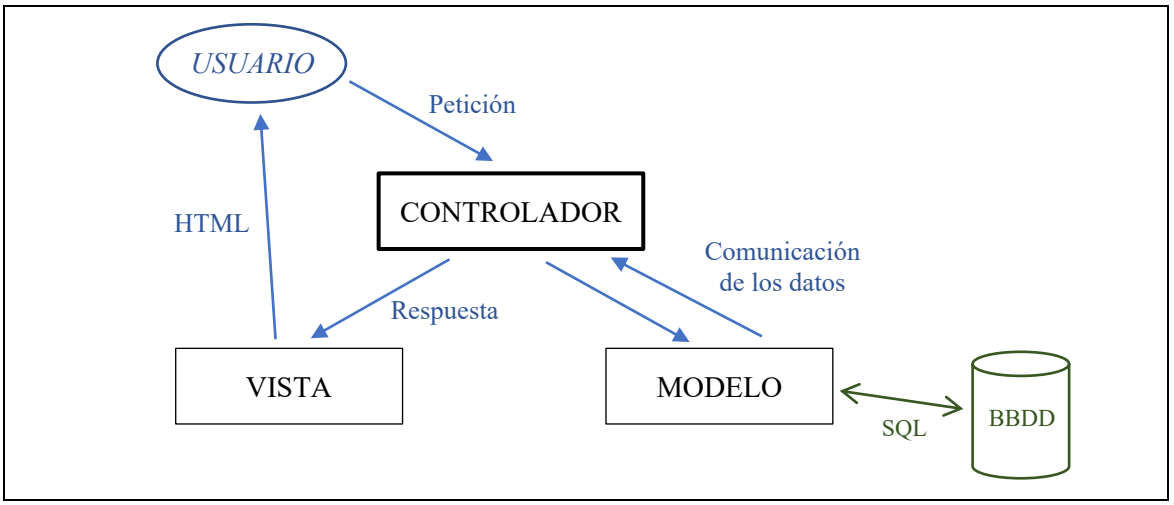

Figura 2. Modelo-Vista-Controlador

Desarrollando un CRUD

Vamos a realizar un CRUD utilizando el patrón MVC. Nos vamos a centrar en la gestión de un cliente, el cual estará almacenado en una base de datos MySQL. La tabla clientes tendrá la siguiente estructura.

Campo	Tipo
id	int(11), PK, autoincremental
nombre	varchar(32)
email	varchar(100)

La estructura de archivos de un proyecto MVC puede estar distribuida de la siguiente forma:

- model
  - bd.php
  - clientes.php
- view
  - css
    - app.css

    - layout
        - header.php
        - menu.php
        - footer.php
    - clientesmantenimiento.php
    - clientes.php
    - app.php
- controller
    - clientes.php
    - app.php
- config.php
- index.php

Archivo de configuración y punto de entrada a la aplicación

El archivo `config.php` contendrá la configuración de nuestra aplicación.

```
define("URLSITE", "http://localhost/crudmvc/");

define("SERVIDOR", "localhost");
define("USUARIO", "root");
define("CONTRASENA", "12345");
define("BASEDATOS", "mvc");
```

Una vez que hemos definido la dirección nuestro sitio web y los valores de acceso a la base de datos, desarrollaremos el punto de entrada de la aplicación `index.php`.

```
<?php
if (session_status() === PHP_SESSION_NONE)
    session_start();

require_once("config.php");
require_once("controller/app.php");
require_once("controller/clientes.php");

$controlador = '';
if(isset($_GET['c'])) :
    $controlador = $_GET['c'];

    $metodo = '';
    if(isset($_GET['m']))
        $metodo = $_GET['m'];

    switch($controlador) :
        case 'clientes' :
            if (method_exists(ClientesControlador, $metodo)) :
                ClientesControlador::{$metodo}();
            else
                ClientesControlador::index();
            endif;

            break;
        default:
            AppControlador::index();
    endswitch;
else :
    AppControlador::index();
endif;
?>
```

Lo primero que realizamos en el requerimiento de la configuración y de los controladores de la aplicación (`controller/app.php`) y clientes (`controller/clientes.php`). Después verificamos que se ha pasado por GET tanto el controlador, en el argumento `c`, como el método del controlador a ejecutar, en el argumento `m`. Si se ha pasado, verificaremos que existe el método en

el controlador `ClientesControlador` mediante la función `method_exists`[90]. En caso contrario, llamaremos al método `index` del controlador de clientes.

Si no han pasado ningún controlador válido, vamos a la página principal de la aplicación, es decir, al método `index` del controlador `AppControlador`.

Modelo

Vamos a analizar el modelo que se encargará de desarrollar todo lo necesario para acceder a la base de datos. Los archivos se guardarán en la carpeta `model`. El primer archivo que codificaremos será el archivo `bd.php` que será el encargado de hacer todo el trabajo con la base de datos.

```
<?php
require once("config.php");

class BD
{
    private $con   = null; // Conexión a la BBDD.
    private $error = '';   // Mensaje de error.

    function __construct()
    {
        $this->error = '';

        try
        {
            // Creamos la conexión.
            $this->con = new PDO('mysql:host=' . SERVIDOR .
                                 ';dbname=' . BASEDATOS .
                                 ';charset=utf8',
                                 USUARIO,
                                 CONTRASENA);

            // Si se logra crear la conexión.
            if ($this->con)
            {
                // Ponemos los atributos para gestionar los errores con excepciones.
                $this->con->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

                // El juego de caracteres será utf-8
                $this->con->exec('SET CHARACTER SET utf8');
            }
        }
        catch (PDOException $e)
        {
            $this->error = $e->getMessage();
        }
    }

    function __destruct()
    {
        // Cerramos la conexión a la BBDD.
        $this->con = null;
    }

    protected function _consultar($query)
    {
        $this->error = '';

        $filas = null;

        try
        {
            // Preparamos la consulta...
            $stmt = $this->con->prepare($query);

            // y la ejecutamos.
            $stmt->execute();

            // Si nos devuelve alguna fila...
```

[90] https://www.php.net/manual/es/function.methos_exists.php

```
            if ($stmt->rowCount() > 0)
            {
                // Creamos el array...
                $filas = array();

                // y lo rellenamos con los datos de la consulta.
                while ($registro = $stmt->fetchObject())
                    $filas[] = $registro;
            }
        }
        catch (PDOException $e)
        {
            $this->error = $e->getMessage();
        }

        // Devolvemos las filas obtenidas de la consulta.
        return $filas;
    }

    protected function _ejecutar($query)
    {
        $this->error = '';

        $filas = 0;

        try
        {
            // Ejecutamos la sentencia y guardamos el número de filas afectadas.
            $filas = $this->con->exec($query);
        }
        catch (PDOException $e)
        {
            $this->error = $e->getMessage();
        }

        // Devolvemos el número de filas afectadas.
        return $filas;
    }

    protected function _ultimoId()
    {
        // Devolvemos el id de la última fila insertada.
        return $this->con->lastInsertId();
    }

    public function GetError()
    {
        // Obtenemos el mensaje del error, si este se produce.
        return $this->error;
    }

    public function Error()
    {
        // Indicamos si ha habido algún error.
        return ($this->error != '');
    }
}
?>
```

Ahora crearemos el modelo para gestionar la tabla clientes de nuestra base de datos.

```
<?php
require_once("bd.php");

class ClientesModelo extends BD
{
    // Campos de la tabla.
    public $id;
    public $nombre;
    public $email;

    public function Insertar()
    {
        $sql = "INSERT INTO clientes VALUES".
               " (default, '$this->nombre', '$this->email')";
```

```
            return $this->_ejecutar($sql);
        }

        public function Modificar()
        {
            $sql = "UPDATE clientes SET" .
                   " nombre='$this->nombre', email='$this->email'" .
                   " WHERE id=$this->id";

            return $this-> ejecutar($sql);
        }

        public function Borrar()
        {
            $sql = "DELETE FROM clientes WHERE id=$this->id";

            return $this->_ejecutar($sql);
        }

        public function Seleccionar()
        {
            $sql = 'SELECT * FROM clientes';

            // Si me han pasado un id, obtenemos solo el registro indicado.
            if ($this->id != 0)
                $sql .= " WHERE id=$this->id";

            $this->filas = $this->_consultar($sql);

            if ($this->filas == null)
                return false;

            if ($this->id != 0)
            {
                // Guardamos los campos en las propiedades.
                $this->nombre = $this->filas[0]->nombre;
                $this->email  = $this->filas[0]->email;
            }

            return true;
        }
    }
    ?>
```

Vista

Vamos a programar las vistas de nuestra aplicación, las cuales las guardaremos en la carpeta `view`. Empezaremos codificando la cabecera `header.php` y el pie `footer.php`, estos dos archivos los guardaremos en la carpeta `view/layout`. Por último, se codifica el archivo `app.css`, el cual guardaremos en la carpeta `view/css`, para darle un poco de estilo junto con Bootstrap a nuestra aplicación.

Primero codificamos la cabecera en el fichero `header.php`.

```
    <!DOCTYPE html>
    <html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <!-- Bootstrap CSS -->
        <link
            href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
            rel="stylesheet"
            integrity=
                 "sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
            crossorigin="anonymous">

        <link rel="stylesheet" type="text/css"
            href="<?php echo URLSITE; ?>view/css/app.css">
        <title>CRUD MVC</title>
    </head>
    <body>
```

```
    <?php require("layout/menu.php"); ?>

    <br>

    <div class="panel">
```

Después el fichero `menu.php`.

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="<?php echo URLSITE; ?>">Inicio</a>
    <button class="navbar-toggler"
            type="button"
            data-bs-toggle="collapse"
            data-bs-target="#navbarSupportedContent"
            aria-controls="navbarSupportedContent"
            aria-expanded="false"
            aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle"
             href="#"
             id="ClientesDropdown"
             role="button"
             data-bs-toggle="dropdown"
             aria-expanded="false">Clientes</a>
          <ul class="dropdown-menu"
              aria-labelledby="clientesDropdown">
            <li><a class="dropdown-item"
                   href="<?php echo URLSITE . '?c=clientes'; ?>">Clientes...</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item"
                   href="<?php echo URLSITE . '?c=clientes&m=facturas'; ?>">
                 Facturas...</a></li>
          </ul>
        </li>
      </ul>
      <span class="navbar-text">
          <a class="nav-item nav-link active"
             href="<?php echo URLSITE . '?c=ayuda'; ?>"> Ayuda</a>
      </span>
    </div>
  </div>
</nav>
```

Y por último, el pie en el fichero `footer.php`.

```
    </div>
</body>
</html>
```

Seguimos con un poco de estilo de nuestra aplicación codificando el fichero `app.css`.

```
.panel
{
   background: white;
   padding: 20px;
   box-shadow: 0px 0px 10px;
}
```

Finalmente codificaremos cada una de las vistas de nuestro CRUD.

Empezamos por la vista de la página principal codificando el fichero `app.php`.

```
<?php require("layout/header.php"); ?>

<h1>CRUD :: PÁGINA PRINCIPAL</h1>

<br/>
```

```
<?php require("layout/footer.php"); ?>
```

Para mostrar los clientes a gestionar codificando el fichero `clientes.php`.

```
<?php require("layout/header.php"); ?>

<h1>CLIENTES</h1>

<br/>

<table class="table table-striped table-hover" id="tabla">
<thead>
<tr class="text-center">
    <th>Id</th>
    <th>Nombre</th>
    <th>Email</th>
    <th></th>
</tr>
</thead>
<tbody>
<?php
if ($clientes->filas) :
    foreach ($clientes->filas as $fila) :
?>
<tr>
    <td style="text-align: right; width: 5%;"><?php echo $fila->id; ?></td>
    <td><?php echo $fila->nombre; ?></td>
    <td><?php echo $fila->email; ?></td>
    <td style="text-align: right; width: 50%;">
        <a href="index.php?c=clientes&m=editar&id=<?php echo $fila->id; ?>">
            <button type="button" class="btn btn-success">Editar</button></a>
        <a href="index.php?c=clientes&m=borrar&id=<?php echo $fila->id; ?>" >
            <button type="button" class="btn btn-danger borrar"
            onclick="return confirm('¿Estás seguro de borrar el registro <?php
                    echo $fila->id; ?>?');">Borrar</button></a>
    </td>
</tr>
<?php
    endforeach;
endif;
?>
</tbody>
<tfoot>
<tr>
    <td colspan="4">
        <a href="index.php?c=clientes&m=nuevo">
        <button type="button" class="btn btn-primary">Nuevo</button>
        </a>
    </td>
</tr>
</tfoot>
</table>

<?php require("layout/footer.php"); ?>
```

A continuación, codificamos la vista para añadir o modificar los clientes en el fichero `clientesmantenimiento.php`. Observar que según el valor de la variable `$opcion` cambiamos el comportamiento del formulario, haciendo que se comporte como un formulario para insertar o bien para modificar.

```
<?php require("layout/header.php"); ?>

<h1>CLIENTES</h1>

<br/>

<h2><?php echo ($opcion == 'EDITAR' ? 'MODIFICAR' : 'NUEVO'); ?></h2>

<form action="<?php echo 'index.php?c=clientes&m=' .
             ($opcion == 'EDITAR' ? 'modificar&id=' . $cliente->id : 'insertar'); ?>"
      method="POST">
  <label for="nombre" class="form-label">Nombre</label>
  <input type="text"
         class="form-control"
```

```
          name="nombre"
          id="nombre"
          value="<?php echo ($opcion == 'EDITAR' ? $cliente->nombre : ''); ?>"
          required/>
  <br/>
  <label for="email" class="form-label">Email</label>
  <input type="text"
          class="form-control"
          name="email"
          id="email"
          value="<?php echo ($opcion == 'EDITAR' ? $cliente->email : ''); ?>"
          required/>
  <br/>
  <button type="submit" class="btn btn-primary">Aceptar</button>
  <a href="<?php echo URLSITE . '?c=clientes'; ?>">
  <button type="button"
           class="btn btn-outline-secondary float-end">Cancelar</button>
  </a>
</form>

<?php require("layout/footer.php"); ?>
```

La ultimo vista que vamos a ver es la de mostrar los errores de nuestra aplicación

```
<?php
if (session_status() === PHP_SESSION_NONE)
    session_start();

require_once("../config.php");
require("layout/header.php");
?>

<br/>
<div class="alert alert-danger" role="alert">
  <h4 class="alert-heading">¡Error!</h4>
  <p>Ha habido un error al realizar la operación:</p>
  <p style="font-style: italic;"><?php echo $_SESSION["CRUDMVC_ERROR"]; ?></p>
  <hr>
  <p class="mb-0">
    <button type="submit" class="btn btn-primary"
            onclick="window.history.back()">Reintentar</button>
    <a href="<?php echo urlsite; ?>"><button type="button"
       class="btn btn-outline-secondary float-end">Cancelar</button></a>
</p>
</div>

<?php require("layout/footer.php"); ?>
```

Controlador

El controlador de nuestra aplicación gestiona las peticiones solicitadas por el usuario. En la función Nuevo, le asignamos a la variable `$opcion` el valor de `NUEVO` , mientras que en la función `Editar` se le asigna el valor `EDITAR`. Con esto controlamos el comportamiento del formulario como vimos anteriormente.

La codificación del controlador se muestra a continuación.

```
<?php
if (session_status() === PHP_SESSION_NONE)
    session_start();

require_once("model/clientes.php");

class ClientesControlador
{
    static function index()
    {
        $clientes = new ClientesModelo();

        $clientes->Seleccionar();

        require_once("view/clientes.php");
    }
```

```
function Nuevo()
{
     $opcion  = 'NUEVO';  // Opción de insertar un cliente.

    require_once("view/clientesmantenimiento.php");
}

function Insertar()
{
    $cliente = new ClientesModelo();

    $cliente->nombre = $_POST['nombre'];
    $cliente->email  = $_POST['email'];

    if ($cliente->Insertar() == 1)
        header("location:" . URLSITE . '?c=clientes');
    else
    {
        $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

        header("location:" . URLSITE . "view/error.php");
    }
}

function Editar()
{
    $cliente = new ClientesModelo();

    $cliente->id = $_GET['id'];

    $opcion = 'EDITAR'; // Opción de modificar un cliente.

    if ($cliente->seleccionar())
        require_once("view/clientesmantenimiento.php");
    else
    {
        $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

        header("location:" . URLSITE . "view/error.php");
    }
}

function Modificar()
{
    $cliente = new ClientesModelo();

    $cliente->id     = $_GET['id'];
    $cliente->nombre = $_POST['nombre'];
    $cliente->email  = $_POST['email'];

    // Aquí hay que tener cuidado, en el caso de que se pulse el botón de aceptar
    // pero no se haya modificado nada, la función modificar devolverá un cero,
    // por eso hay que comprobar que no hay error.
    if (($cliente->Modificar() == 1) || ($cliente->GetError() == ''))
        header("location:" . URLSITE. '?c=clientes');
    else
    {
        $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

        header("location:" . URLSITE . "view/error.php");
    }
}

function Borrar()
{
    $cliente = new ClientesModelo();

    $cliente->id = $_GET['id'];

    if ($cliente->Borrar() == 1)
        header("location:" . URLSITE. '?c=clientes');
    else
    {
        $_SESSION["CRUDMVC_ERROR"] = $cliente->GetError();

        header("location:" . URLSITE . "view/error.php");
    }
```

```
        }
    }
```

Conclusión

Hemos visto una aplicación sencilla, básica y claramente mejorable, de un CRUD utilizando el patrón MVC. El objetivo es entender cómo funciona este patrón de cara al desarrollo de aplicaciones PHP con frameworks[91] como Laravel, Symfony , Zend, etc.

Le hemos añadido algo de vistosidad utilizando Bootstrap y gestionamos el error a mostrar mediante sesiones.

Ejercicios

115. Amplía el CRUD para que el cliente tenga: apellidos, contraseña, dirección, cp, población, provincia y fecha de nacimiento.

116. Amplia el CRUD para gestionar las facturas de un cliente[92]. Las facturas tendrán la siguiente estructura:

Campo	Tipo
id	int(11), PK, autoincremental
cliente_id	int(11)
numero	int(11)
fecha	datetime

117. Amplía el CRUD para gestionar las líneas de una factura. Las líneas tendrán la siguiente estructura:

Campo	Tipo
id	int(11), PK, autoincremental
factura_id	int(11)
referencia	int(11)
descripcion	datetime
cantidad	decimal(10, 3)
precio	decimal(10,2)
iva	decimal(5,2)
importe	decimal(10, 2)

En la tabla anterior el campo "importe" que será el resultado de la siguiente operación:

importe = cantidad * precio * (1 + iva / 100.0)

[91] https://en.wikipedia.org/wiki/Category:PHP_frameworks

[92] Utilizar la función parse_url, https://www.php.net/manua/es/function.parse-url.php, para saber qué controlador hay que seleccionar: