PHP es un lenguaje de desarrollo web en el lado del servidor. De hecho, es el lenguaje de desarrollo web en el lado del servidor más ampliamente utilizado a día de hoy, y aproximadamente el 80% de las páginas que podamos encontrar en Internet lo utilizan. De ahí la importancia de conocerlo y saberlo utilizar.

1. Características principales de PHP

PHP es un lenguaje de *script* del lado del servidor. Un lenguaje de *script* es básicamente un lenguaje de programación interpretado o ejecutado "al vuelo". Es similar a otros lenguajes de *script* en el lado del servidor, como JSP o ASP.NET.

Además, PHP es un lenguaje multiparadigma. Podemos desarrollar aplicaciones siguiendo el paradigma estructurado (bucles y condiciones), modular (funciones) y orientado a objetos. Tiene una sintaxis bastante particular, y diferente a muchos otros lenguajes. Como lenguaje de servidor, a menudo lo utilizaremos intercalado dentro del contenido de una página HTML.

Podemos encontrar gran variedad de sitios web realizados con PHP: tiendas virtuales en las que poder crearnos un carro de la compra y pagar sobre una plataforma de pago de una entidad bancaria, cursos virtuales en plataformas Moodle (CMS hecho en PHP), o páginas web más o menos complejas realizadas directamente en PHP, o con algún gestor de contenidos en PHP como Wordpress o Joomla.

Entre sus principales características, aparte de su popularidad, podemos citar las siguientes:

- Es open source, por lo que podemos acceder a su código y usarlo, modificarlo o distribuirlo.
- Fácil de aprender, si se compara con otros lenguajes de programación como C o Perl.
- Multiplataforma, se puede ejecutar en diversos sistemas operativos. Esta es una diferencia importante con otras tecnologías como .NET (exclusiva de Windows), aunque hay otros lenguajes de servidor que también son multiplataforma, como Java/JSP.
- Permite acceder a diversos sistemas de bases de datos (MySQL, Oracle, PostgreSQL...)
- Admite diversas librerías y entornos para implementar sistemas de comercio electrónico, envío de e-mails, frameworks de desarrollo, etc.

2. Recursos necesarios para trabajar con PHP

Para poder trabajar con PHP necesitamos un **servidor web** que dé soporte a este lenguaje. Un *servidor web* es una herramienta que permite escuchar peticiones (locales o remotas) que llegan a un puerto del ordenador (normalmente el puerto 80, que es el puerto por defecto), y las procesa mediante algún recurso interno (en nuestro caso, las páginas PHP que alojaremos en dicho servidor), para ofrecer una respuesta a dicha petición. Por ejemplo, podemos solicitar un listado de libros de una base de datos, y en este caso la página PHP que

reciba la petición deberá encargarse de conectar a la base de datos, obtener el listado y enviarlo a quien realizó la petición (cliente) en el formato esperado.

Existen varios servidores web que podemos utilizar para trabajar con PHP, siendo los más populares **Apache** y **Nginx**. Además, necesitaremos instalar el propio lenguaje **PHP**, y normalmente será necesario acceder a algún tipo de base de datos.

2.1 Utilizando XAMPP

Para aunar estos tres problemas (servidor web, PHP y base de datos) en una única solución, una alternativa bastante habitual consiste en instalar una herramienta llamada **XAMPP**. Este software incorpora:

- Servidor web Apache
- Servidor de base de datos MySQL
- Lenguaje PHP

Una de las ventajas que ofrecen es que, además de instalar Apache, PHP y MySQL y dejarlo todo integrado, nos proporciona un cliente web llamado **phpMyAdmin** para poder administrar las bases de datos desde Apache. Esto nos vendrá bien para crear o importar las bases desde una interfaz gráfica.

En cuanto a la **versión requerida**, depende un poco del tipo de aplicaciones que queramos desarrollar. Podemos descargar la última versión disponible desde la web oficial e instalarla dependiendo de nuestro sistema operativo.

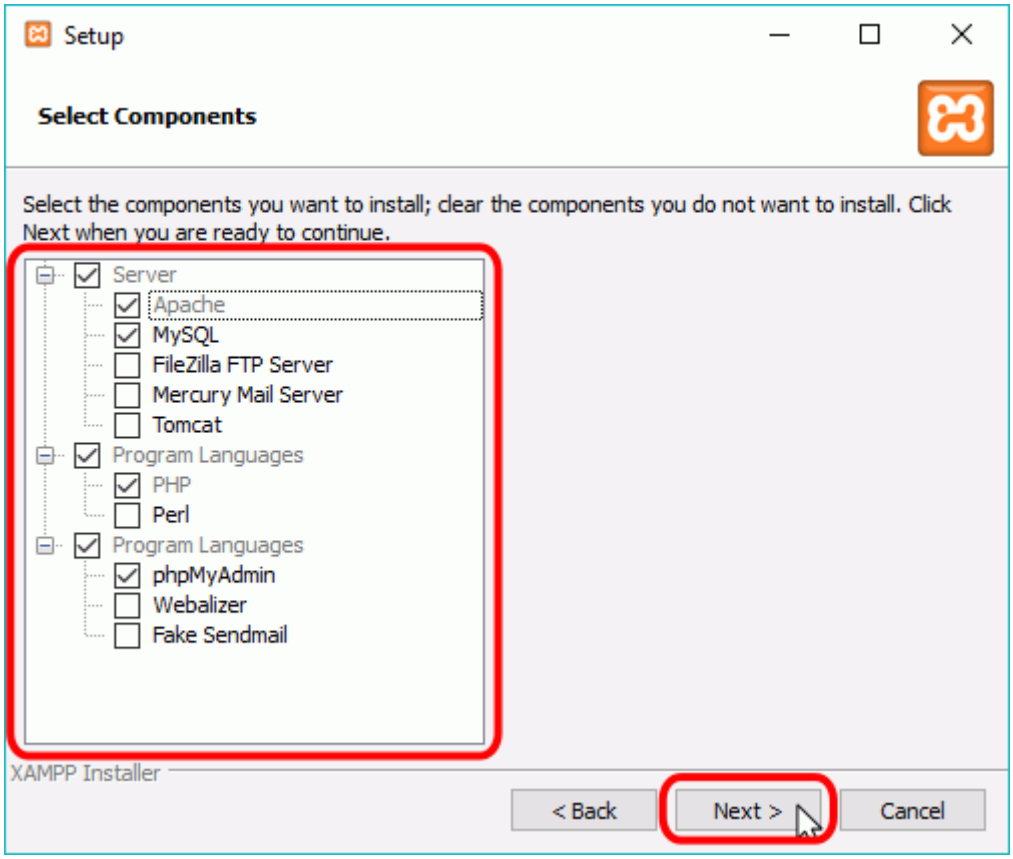
Linux

En el caso de Linux, debemos dar permisos de ejecución y ejecutar el archivo *.run* que descarguemos desde algún terminal, con permisos de administrador (*sudo*). Suponiendo que el archivo se llame *xampp-linux-x64-8.1.2-installer.run*, por ejemplo, los pasos son los siguientes (desde la carpeta donde lo hemos descargado):

```
sudo chmod +x xampp-linux-x64-8.1.2-installer.run
sudo ./xampp-linux-x64-8.1.2-installer.run
```

Windows y MacOSX

En el caso de **Windows** o **Mac OSX** simplemente hay que lanzar el instalador y seguir los pasos, eligiendo las opciones que nos interese instalar (al menos, Apache, MySQL y PHP), si nos dan a elegir. Así es como podemos dejarlo en el caso de Windows, por ejemplo:

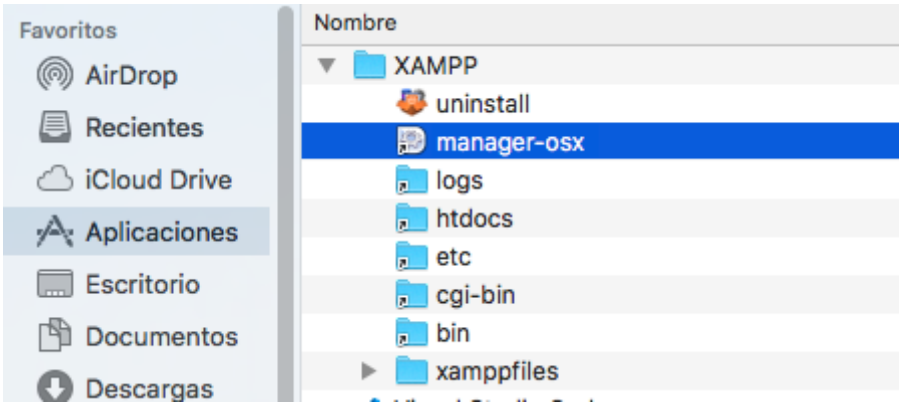


El manager de XAMPP

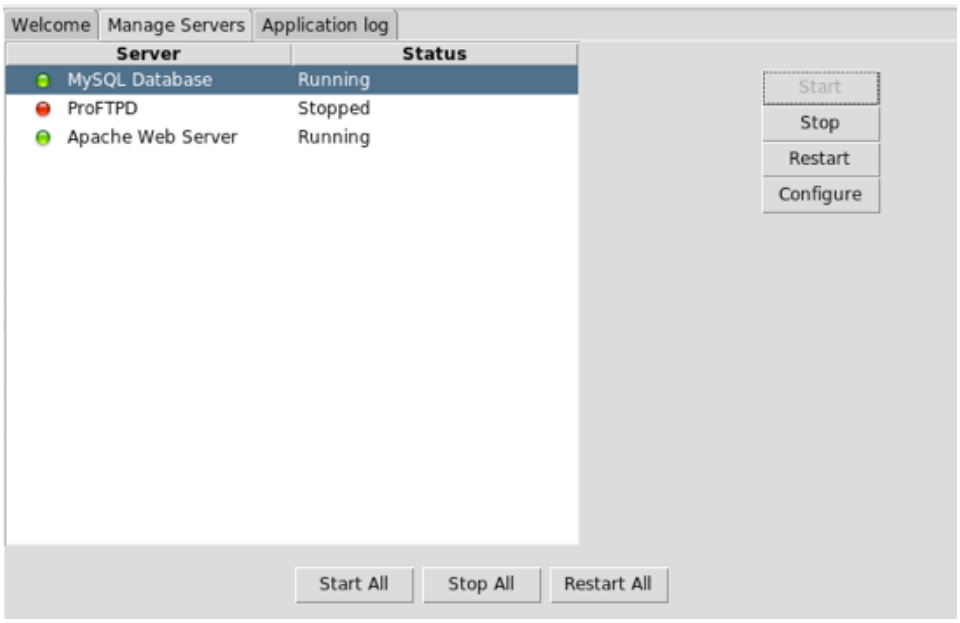
XAMPP proporciona una herramienta *manager* o *panel de control* que nos permite gestionar en todo momento los servicios activos.

En el caso de **Linux** se encuentra en **/opt/lampp/manager-linux-x64.run**. Podemos acceder a la carpeta desde el terminal para ejecutarlo (con permisos de superusuario), o bien crear algún acceso directo en otra ubicación que nos resulte más cómoda.

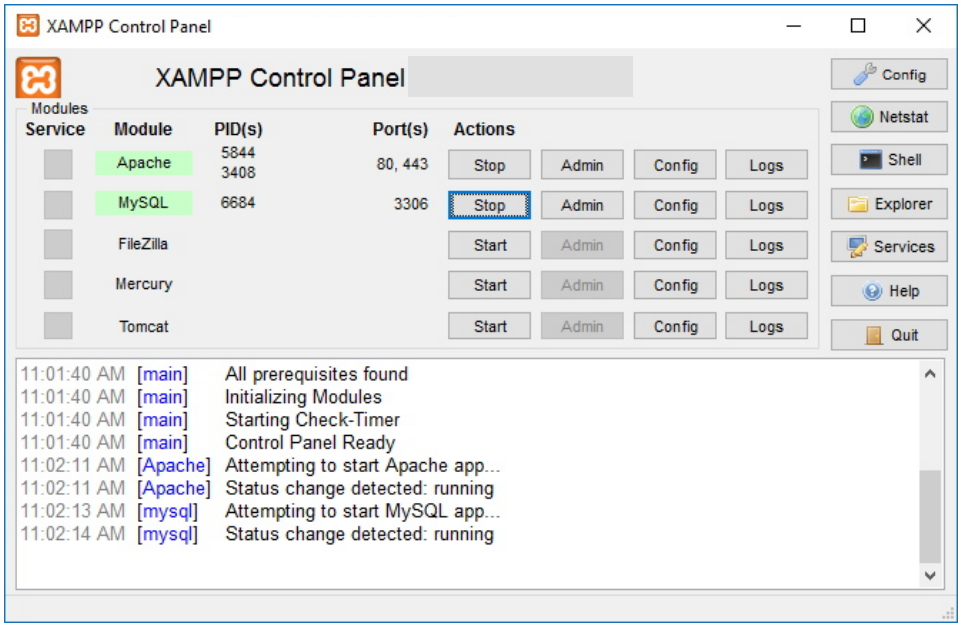
En el caso de **Windows**, dicho manager está en la carpeta de instalación (típicamente *C:\xampp*), en un archivo llamado *xampp-control.exe*, que podemos ejecutar. En el caso de **Mac OSX**, se habrá creado un acceso en la sección de *Aplicaciones* para poder poner en marcha este manager.



El manager nos permitirá lanzar o detener cada servidor. Para las pruebas que haremos deberemos tener iniciados tanto Apache como MySQL. En Linux y Mac OS X tendrá una apariencia como ésta aproximadamente:



En el caso de Windows la apariencia es algo diferente, aunque igualmente funcional:



Por defecto, Apache estará escuchando en el puerto 80 (o 443 para conexiones SSL), y MySQL en el 3306. Podemos modificar estos puertos en los respectivos archivos de configuración ("*httpd.conf*" y "*my.cnf*"), dentro de las carpetas de la instalación de XAMPP (la ubicación concreta de estos archivos varía entre versiones y entre sistemas operativos).

> **IMPORTANTE**: en el caso de Windows, es posible que tengamos que dar permisos de *Control total* al archivo *xampp-control.ini*, que se creará en la carpeta de instalación de XAMPP la primera vez que ejecutemos el *manager*. Esto deberá hacerse para evitar el mensaje de error que puede aparecer al intentar acceder a ese archivo sin permisos por parte de la aplicación.

Ubicación de las webs

La carpeta por defecto donde debemos ubicar nuestras webs es la subcarpeta **htdocs** dentro de la carpeta de instalación de XAMPP (típicamente *C:\xampp* en Windows, u */opt/lampp* en Linux). Ahí deberemos colocar las páginas y carpetas que necesitemos, incluso podemos crear una carpeta para cada aplicación.

En realidad, podemos configurar Apache para admitir otras rutas distintas, y alojar así nuestros distintos proyectos en diversas carpetas a nuestra elección. Pero esto queda fuera del alcance de este curso, así que los ejemplos y ejercicios que probaremos los crearemos en carpetas independientes dentro de la carpeta por defecto.

> **Ejercicio 1:**
>
> Descarga e instala XAMPP en tu ordenador. Abre el *manager* y pon en marcha Apache y MySQL para comprobar que arrancan satisfactoriamente. Intenta acceder a la página de inicio (típicamente *http://localhost*, si no has tenido que cambiar el puerto por defecto).
>
> Después, crea la carpeta *prueba* dentro de la carpeta *htdocs* y añade dentro un archivo *index.html* con un mensaje de bienvenida. Prueba a acceder a dicha página (con Apache en marcha) accediendo a *http://localhost/prueba/index.html*.

3. Mi primer programa PHP

Como comentábamos antes, PHP es un lenguaje con una sintaxis bastante particular. Normalmente los archivos tienen extensión *.php* (aunque más adelante veremos que hay otras alternativas), y dentro de ellos, el código comienza con la expresión `<?php`. Todo lo que esté englobado entre esta expresión y la de cierre `?>` es código PHP, y podemos intercalar tantos bloques de este tipo como queramos en nuestras páginas.

Por ejemplo, esta página `bienvenida.php` carga un contenido HTML y, en medio, define un fragmento de código con un texto de bienvenida que muestra por pantalla:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <h1>Página de bienvenida</h1>
    <?php
        echo "<p>Bienvenido a esta página</p>";
    ?>
</body>
</html>
```

> **Ejercicio 2:**
>
> En la misma carpeta *htdocs/prueba* del ejercicio anterior, crea una página llamada `informacion.php` y, utilizando la instrucción `echo` de PHP que hemos visto en el ejemplo anterior, muestra algo de información sobre tí, generando algunos párrafos con formato HTML: nombre, aficiones, estudios, etc. Prueba a cargar la página (con el servidor Apache en marcha) y comprueba que el contenido se genera con el formato esperado.

3.1. Esquema de funcionamiento general

¿Cómo hace un servidor web como Apache para, a través de PHP, procesar una petición y generar un contenido para el cliente? Los pasos que se siguen son:

1. El cliente realiza una petición de un recurso PHP. Típicamente consiste en solicitar una página PHP desde el navegador, a través de un enlace o formulario. Por ejemplo, un formulario que envía sus datos a *pagina.php*
2. El servidor recibe la petición, toma el archivo *pagina.php* y lo interpreta, es decir, lanza el intérprete PHP para ejecutar el código PHP de la página
3. Se sustituye el código PHP ejecutado por el código HTML que se ha generado en su lugar
4. Se envía el código HTML generado como respuesta al cliente.

Por ejemplo, al recibir una petición a la página *bienvenida.php* del ejemplo anterior, se lanza el intérprete PHP y se genera el siguiente contenido HTML como resultado, que es lo que se devuelve al navegador:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <h1>Página de bienvenida</h1>
    <p>Bienvenido a esta página</p>
</body>
</html>
```

Como podemos ver e imaginar, en el navegador NO se puede ver el código PHP del servidor, ya que éste ha sido procesado, interpretado y sustituido por el HTML correspondiente en el servidor.

Elementos básicos del lenguaje

Como hemos visto en el documento anterior, la sintaxis básica para escribir código PHP consiste en un bloque como el siguiente:

```
<?php
    ... código PHP ...
?>
```

Dentro de los símbolos `<?php` (de apertura) y `?>` (de cierre) pondremos las instrucciones que consideremos. Todo esto puede ocupar una sola línea (para instrucciones simples y cortas) o dividir el código en varias líneas, una por instrucción.

1. Sintaxis básica y comentarios

Dentro de cada bloque de código PHP que escribamos en una página, debemos tener en cuenta que:

- Cada instrucción en PHP termina siempre en un punto y coma (`;`) que la separa de la siguiente, aunque estén en líneas separadas.
- Los comentarios en PHP se pueden poner de diversas formas.

```
/* Comentario que puede ocupar varias líneas */
// Comentario de una línea
# Otro ejemplo de comentario de una línea
```

Veamos a continuación un ejemplo sencillo de código PHP, embebido dentro de una página HTML. En este caso, creamos unas variables en PHP que almacenan un nombre y un año, y luego mostramos esas variables entre el contenido HTML propiamente dicho:

```
<!DOCTYPE html>
<html lang="es">
<head>
    ...
</head>
<body>
    <h1>Página de prueba en PHP</h1>
    <?php
        // Variables para almacenar el nombre y el año actual
        $nombre = "Nacho Iborra";
        $anyo = 2014;
    ?>
    <p>El autor de esta página es <?php echo $nombre; ?> y está
    realizada en el año <?php echo $anyo; ?>.</p>
...
```

Observa cómo podemos incluir código PHP en cualquier zona de la página. En la primera, hemos definido dos variables, `$nombre` y `$anyo`, y después, con instrucciones cortas, las hemos mostrado en otras zonas de la página (instrucción `echo`).

Lo que hace el servidor cuando se le solicita esta página es procesarla, detectar el código PHP, ejecutarlo y sustituirlo en la página por el código HTML que éste genera (en el ejemplo anterior, mostrar el valor de las variables en los lugares correspondientes).

Opcionalmente, en el caso de que sólo tengamos instrucciones `echo`, podemos sustituir la estructura `<?php ... ?>` por la estructura `<?= ... ?>` y ahorrarnos la instrucción *echo*. Así, el párrafo que se muestra en el ejemplo anterior lo podríamos poner así:

```
<p>El autor de esta página es <?= $nombre; ?> y está
realizada en el año <?= $anyo; ?>.</p>
```

También podemos utilizar indistintamente la instrucción `print` en lugar de `echo` para mostrar información por pantalla.

```
<p>El autor de esta página es <?php print $nombre; ?> y está
realizada en el año <?php print $anyo; ?>.</p>
```

> **Ejercicio 1:**
>
> Crea una carpeta llamada **ejercicios1** en tu carpeta de documentos de Apache. En esta carpeta guardarás este ejercicio y los siguientes, ya que serán muchos y así evitamos llenar la carpeta de documentos de demasiadas subcarpetas con ejercicios cortos.

> Para este ejercicio, crea un documento en esta carpeta llamado **info_basica.php**, similar al del ejemplo anterior, pero mostrando tu nombre y tu año de nacimiento usando variables. Es decir, crearás dos variables para almacenar estos dos datos, y los mostrarás en una frase que diga "Me llamo XXXX y nací en el año YYYY". Prueba la página en un navegador y echa un vistazo al código fuente, intentando detectar qué contenidos HTML se han generado desde PHP.

2. Variables y tipos de datos

Como en todo lenguaje de programación, las variables en PHP nos van a servir para almacenar información, de manera que, además de tenerla disponible, podemos modificarla o cambiarla por otra durante el tiempo de ejecución de la aplicación web.

Como hemos visto en el ejemplo anterior, las variables en PHP se definen mediante el símbolo del dólar (`$`) seguido de una serie de letras, números o caracteres de subrayado, aunque no pueden empezar por número. Ejemplos de nombres de variables válidos son: `$nombre`, `$primer_apellido`, `$apellido2`... En las variables, se distinguen mayúsculas de minúsculas, y no hace falta declararlas (es decir, no se indica de qué tipo son, como en lenguajes como C o Java, ni se les reserva memoria de antemano).

Veamos algunos ejemplos de uso de variables:

```php
<?php
    $edad = 36;
    $nombre = "Nacho";
    ...
    echo $nombre;
    echo $edad;
?>
```

2.1. Comprobar el estado de las variables

Es posible que, en algún momento de la ejecución del programa, una variable no tenga un valor definido, o queramos eliminar el valor que tiene. Para ello tenemos algunas instrucciones útiles:

- `unset($variable)` permite borrar la variable (como si no la hubiéramos creado)
- `isset($variable)` permite comprobar si una variable existe
- `empty($variable)` permite comprobar si una variable está vacía, es decir, no tiene un valor concreto asignado.

```php
<?php
    $dato="Hola";
    unset($dato);       // La variable dato deja de existir
    $dato = "";
    echo empty($dato);  // Diría que es cierto, porque $dato está vacía
?>
```

2.2. Tipos de datos

Las variables almacenan datos, y esos datos son de un tipo concreto. Por ejemplo, pueden ser números, o textos. En concreto, PHP soporta los siguientes tipos de datos básicos:

- **Booleanos**: datos que sólo pueden valer verdadero o falso (en PHP se representan con las palabras TRUE y FALSE, respectivamente, en mayúsculas).
- **Enteros**: números sin decimales. Los podemos representar en formato decimal (el normal, por ejemplo 79), formato octal (poniendo un 0 delante, por ejemplo 0233) o hexadecimal (poniendo 0x delante, por ejemplo 0x1A3).
- **Reales**: números con decimales. La parte decimal queda separada de la entera con un punto. Por ejemplo, 3.14. También podemos usar notación científica, como por ejemplo 1.3e3, que equivale a $1.3 \cdot 10^3$.
- **Textos**: se pueden representar entre comillas simples o dobles. Si los representamos con comillas dobles, podemos intercalar variables dentro.

Veamos algunos ejemplos de cada tipo:

```php
<?php
    $edad = 20;
    $esMayorEdad = TRUE;
    $nota = 9.5;
    $nombre = "Juan Pérez";

    echo "El alumno $nombre tiene una nota de $nota";
?>
```

Además, en PHP podemos manejar otros tipos de datos algo más complejos, que son:

- **Array**: conjuntos de datos
- **Object**: para programación orientada a objetos, almacena instancias de clases
- **Resource**: para recursos externos, como una conexión a una base de datos
- **null**: es un valor especial para darle a variables que, en un momento dado, no tengan un valor concreto. Así, quedan vacías, sin valor.

2.2.1. Conversiones entre tipos de datos

Si tenemos un dato de un tipo y lo queremos convertir a otro, se tienen una serie de funciones que nos permiten hacer estas conversiones.

- **intval** sirve para convertir un valor (en formato cadena, o real) a entero.
- **doubleval** sirve para convertir un valor a número real.
- **strval** sirve para convertir un valor a una cadena de texto.

Es habitual usarlas cuando le pedimos datos al usuario en un formulario. Si por ejemplo le pedimos que escriba su edad en un cuadro de texto, este valor se envía como tipo cadena de texto, no como un número, y luego tendríamos que convertirlo a número entero. Realmente, estas conversiones son automáticas con las últimas versiones de PHP, pero conviene saber que estas funciones existen para poderlas utilizar si es el caso. Veamos otro ejemplo más sencillo, con una variable cadena de texto que convertimos al entero correspondiente:

```
<?php
    $textoEdad='21';
    $edad = intval($textoEdad);
?>
```

2.3. Constantes

Hemos visto que las variables son datos cuyo valor puede cambiar a lo largo de la ejecución de la aplicación. A veces nos puede interesar almacenar otros datos que no queramos que cambien a lo largo del programa. Por ejemplo, si almacenamos el número *pi*, ese valor siempre va a ser el mismo, no lo vamos a cambiar.

Para definir constantes en PHP se utiliza la función `define`, y entre paréntesis pondremos el nombre que le damos a la constante (entre comillas), y el valor que va a tener, separados ambos datos por coma. Después, para utilizar la constante más adelante, usamos el nombre que le hemos dado, pero sin las comillas. Veamos este ejemplo que calcula la longitud de una circunferencia:

```
<?php
    define('PI', 3.1416);
    $radio = 5;
    $longitud = 2 * PI * $radio;

    echo "La longitud de la circunferencia es $longitud";
?>
```

Aunque no es obligatorio, sí es bastante convencional que las constantes tengan todo su nombre en mayúsculas, para distinguirlas a simple vista de las variables (aunque, además, las variables en PHP empiezan por un dólar, y las constantes no).

> **Ejercicio 2:**

> Crea una página en la carpeta de ejercicios llamada **area_circulo.php**. En ella, crea una variable `$radio` y ponle el valor 3.5. Según esa variable, calcula en otra variable el área del círculo (PI * radio^2, deberás definir la constante PI), y muestra por pantalla el texto "El área del círculo es XX.XX", donde XX.XX será el resultado de calcular el área.

2.4. Operaciones

Podemos realizar distintos tipos de operaciones con los datos que manejamos en PHP: aritméticas, comparaciones, asignaciones, etc. Veremos los operadores que podemos utilizar en cada caso.

Operaciones aritméticas

Son las operaciones matemáticas básicas (sumar, restar, multiplicar...). Los operadores para llevarlas a cabo son:

Operador	Significado
`+`	Suma
`-`	Resta
`*`	Multiplicación
`/`	División
`%`	Resto de división
`.`	Concatenación (para textos)
`++`	Autoincremento
`--`	Autodecremento

El operador . sirve para concatenar o enlazar textos, de forma que podemos unir varios en una variable o al sacarlos por pantalla:

```
$edad = 36;
$nombre = "Nacho";
$texto = "Hola, me llamo " . $nombre . " y tengo " . $edad . " años.";
// En este punto, $texto vale "Hola, me llamo Nacho y tengo 36 años."
$edad++;
echo "El usuario se llama " . $nombre;
```

Operaciones de asignación

Permiten asignar a una variable un cierto valor. Se tienen los siguientes operadores:

Operador	Significado
`=`	Asignación simple
`+=`	Autosuma
`-=`	Autoresta
`*=`	Automultiplicación
`/=`	Autodivisión
`%=`	Autoresto
`.=`	Autoconcatenación

La asignación simple ya la hemos visto en ejemplos previos, y sirve simplemente para darle un valor a una variable.

```
$dato = 3;
```

Los operadores de Auto... afectan a la propia variable, sumándole/restándole/... etc. un valor. Por ejemplo, si hacemos algo como:

```
$dato *= 5;     // Dato = 15
```

Operaciones de comparación

Estas operaciones permiten comparar valores entre sí, para ver cuál es mayor, o si son iguales, entre otras cosas. En concreto, los operadores disponibles son:

Operador	Significado
`==`	Igual que
`===`	Idéntico a (igual y del mismo tipo)
`!=, <>`	Distinto de
`!==`	No idéntico
`<`	Menor que
`<=`	Menor o igual que
`>=`	Mayor o igual que

Lo que se obtiene con estas comparaciones es un valor booleano (es decir, verdadero o falso). Por ejemplo, 5 != 3 sería verdadero, y 4 <= 1 sería falso.

Además, en versiones recientes de PHP se han añadido algunos operadores adicionales, como el operador *nave espacial* y el operador de comprobación de nulos.

- El operador de nave espacial `<=>` compara dos datos y devuelve -1 si el primero es menor, 1 si es mayor o 0 si son iguales. Tiene un funcionamiento similar a la función *compareTo* que existe en otros lenguajes como Java o C#.

```
$a = 3;
$b = 5;
echo $a <=> $b;     // -1
```

- Por su parte, el operador de comprobación de nulo `??` se emplea para determinar si una variable tiene valor nulo, y ofrecer una alternativa en ese caso:

```
// Mostrará "No existe el dato" si $dato es nulo
echo $dato ?? 'No existe el dato';
```

Operaciones lógicas

Estas operaciones permiten enlazar varias comprobaciones simples, o cambiarles el sentido, según el operador. En concreto tenemos estos operadores lógicos en PHP:

Operador	Significado
`and , &&`	Operador AND (Y)
`or, ||`	Operador OR (O)
`xor`	Operador XOR
`!`	Negación (NO)

Ejemplos:

```
echo $n1 >= 0 && $n2 >= 0;
echo $n1 >= 0 || $n2 >= 0;
echo $n1 >= 0 xor $n2 >= 0;
```

El operador de negación invierte el sentido de una comprobación (si era verdadera, la vuelve falsa, y viceversa). Por ejemplo, si queremos ver si una edad no es mayor de edad, podríamos ponerlo así:

```
echo !($edad >= 18);
```

Precedencia de operadores

¿Qué ocurre si tenemos varios operadores de distintos tipos en una misma expresión? PHP sigue un orden a la hora de evaluarlos:

1. Toda expresión entre paréntesis
2. Autoincrementos y autodecrementos
3. Negaciones y cambios de signo
4. Multiplicaciones, divisiones y restos
5. Sumas, restas y concatenaciones
6. Comparaciones
7. Operaciones lógicas (`&&` , `||`)
8. Asignaciones

Existen, además, otros operadores que no hemos visto aquí, como operadores de bits, conversiones tipo cast, operador ternario... pero no son tan habituales ni importantes.

> **Ejercicio 3:**
>
> Intenta predecir qué resultado va a sacar por pantalla cada instrucción echo de este código PHP. Luego podrás comprobar si estabas en lo cierto poniendo el código en una página y probándolo en un navegador.

```
<?php
    $num1 = 3;
    $num2 = 5;
    $num3 = 8;
    $num1 *= 4;

    echo $num1;
    echo $num1 <= $num2;
    echo $num3 > $num1 and $num3 > $num2;
    echo $num3 > $num1 or $num3 > $num2;
    echo $num1 > $num2 xor $num1 > $num3;
    $num3--;
    echo $num3;
    $num3 += $num1;
    echo $num3;
?>
```

2.5. Control de flujo

2.5.1. Estructuras selectivas

A veces nos interesa realizar una operación si se cumple una determinada condición y no hacerla (o hacer otra distinta) si no se cumple esa condición. Por ejemplo, si está vacía una variable querremos hacer una cosa, y si no lo está, hacer otra. Para decidir entre varios caminos a seguir en función de una determinada condición, al igual que en otros lenguajes como Javascript, Java o C, se utiliza la estructura **if..else**:

```
if (condicion)
{
    instruccion1a;
    instruccion2a;
    ...
}
else
{
    instruccion1b;
    instruccion2b;
    ...
}
```

Si queremos elegir entre más de dos caminos, podemos utilizar la estructura *if..elseif.. elseif..* y poner una condición en cada if para cada camino. Por ejemplo, imaginemos que tenemos una variable edad donde almacenaremos la edad del usuario. Si el usuario no llega a 10 años le diremos que no tiene edad para ver la web. Si no llega a 18 años, le diremos que aún es menor de edad, pero puede ver la web, y si tiene más de 18 años le diremos que está todo correcto:

```
$edad = ...
if ($edad < 10)
{
    echo "No tienes edad para ver esta web";
}
elseif ($edad < 18)
{
    echo "Aún eres menor de edad, pero puedes acceder a esta web";
}
else
{
    echo "Todo correcto";
}
```

> **Ejercicio 4:**
>
> Crea una página llamada **prueba_if.php** en la carpeta de ejercicios del tema. Crea en ella dos variables llamadas `$nota1` y `$nota2`, y dales el valor de dos notas de examen cualesquiera (con decimales si quieres). Después, utiliza expresiones *if..else* para determinar qué nota es la mayor de las dos.

> **Ejercicio 5:**
>
> Modifica el ejercicio anterior añadiendo una tercera nota `$nota3`, y determinando cuál de las 3 notas es ahora la mayor. Para ello, deberás ayudarte esta vez de la estructura *if..elseif..else*.

También disponemos de la estructura **switch..case**, similar a otros lenguajes, para elegir entre el conjunto de valores que puede tomar una expresión:

```
switch($variable)
{
    case 0:
        echo "La variable es 0";
        break;
    case 1:
        echo "La variable es 1";
        break;
    default:
        echo "La variable es otra cosa";
}
```

2.5.2. Estructuras repetitivas o bucles

Para poder repetir código o recorrer conjuntos de datos, al igual que en otros lenguajes de programación, disponemos de algunas estructuras repetitivas o bucles.

Una de ellas es la estructura **for**. Supongamos que tenemos una variable `$lista` con una lista de elementos. Si queremos recorrerla, tenemos dos opciones: la primera es utilizar una variable que vaya desde el principio de la lista (posición 0) hasta el final (indicado por la función `count`):

```
for ($i = 0; $i < count($lista); $i++)
{
    echo $lista[i];         // Muestra el valor de cada elemento
}
```

La segunda alternativa es utilizar una variable que sirva para almacenar cada elemento de la lista. Para esta opción usamos la estructura **foreach**:

```
foreach ($lista as $elemento)
{
    echo $elemento;         // Muestra el valor de cada elemento
}
```

Además, existe una tercera forma de repetir un conjunto de instrucciones: la estructura **while**. Pondremos entre paréntesis una condición que debe cumplirse para que las instrucciones entre las llaves se repitan. Este código cuenta del 1 al 5:

```
$numero = 1;
while ($numero <= 5)
{
    echo $numero;
    $numero++;
}
```

Una variante de la estructura while es la estructura **do..while**, similar a la anterior, pero donde la condición para terminar el bucle se comprueba al final:

```
$numero = 1;
do
{
    echo $numero;
    $numero++;
} while ($numero <= 5);
```

> **Ejercicio 6:**
>
> Crea una página llamada **contador.php** en la carpeta de ejercicios del tema. Utiliza una estructura *for* para contar los números del 1 al 100 (separados por comas), y luego una estructura *while* para contar los números del 10 al 0 (una cuenta atrás, separada por guiones). Al final debe quedarte algo como esto:

```
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...
10-9-8-7-6-5-4-3-2-1-0
```

2.6. Intercalar control de flujo y HTML

Podemos intercalar bloques de código PHP y bloques HTML, incluso cortando bloques de control de flujo para introducir el código HTML. Por ejemplo:

```
<?php
if ($edad < 10=)
{
?>
   <div class="menor">Eres demasiado pequeño para entrar a esta web</div>
<?php }  elseif ($edad < 18) {  ?>
   <div class="mediano">No eres mayor de edad, pero puedes entrar</div>
<?php } else {  ?>
   <div class="mayor">Todo correcto. Bienvenido</div>
<?php } ?>
```

Sería equivalente a haber hecho un solo bloque PHP que, con instrucciones echo, sacara el contenido HTML, pero a veces es más cómodo poner el HTML directamente.

> **Ejercicio 7:**
>
> Modifica el ejercicio anterior y añádele algún *h1* y párrafos explicativos a la página, fuera del código PHP, explicando lo que se va a hacer. Por ejemplo, que te quede algo así:

Contadores

Este contador va del 1 al 100:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15...

Este otro va del 10 al 0:

10-9-8-7-6-5-4-3-2-1-0

Uso de funciones

Al igual que en otros lenguajes de programación (como por ejemplo JavaScript), las funciones en PHP nos van a permitir encapsular un conjunto de instrucciones bajo un nombre, y poderlo ejecutar en bloque cada vez que lo necesitemos, simplemente utilizando el nombre que le hayamos puesto a la función.

1. Definición de funciones

Las funciones en PHP se definen con la palabra `function` (como en JavaScript), seguida del nombre de la función y unos paréntesis. A diferencia de otros lenguajes, en PHP no se distinguen mayúsculas y minúsculas en los nombres de funciones (sí se distinguen en los nombres de variables). Veamos un ejemplo:

```
function saluda()
{
    echo "Hola, buenas";
}
```

En este caso, la función muestra un mensaje de saludo. Podríamos usarla en cualquier lugar de nuestra página PHP con:

```
saluda();
```

y mostraría en la página web el texto "Hola, buenas" en el lugar donde la hubiéramos colocado.

Notar que estos dos fragmentos de código pueden ir en bloques PHP diferentes. Por ejemplo, podemos definir las funciones al inicio del código PHP, y luego llamarlas después:

```
<?php
function saluda()
{
    echo "Hola, buenas";
}
?>
<!DOCTYPE html>
...
<?php
saluda();
?>
```

2. Uso de parámetros

Como en otros lenguajes, las funciones PHP también admiten unos datos entre los paréntesis, llamados *parámetros*, que sirven para proporcionar datos a la función desde fuera, al llamarla. Por ejemplo, esta función suma los dos datos (numéricos) que se le pasan como parámetros:

```
function suma($a, $b)
{
    echo $a + $b;
}
```

Y para utilizarla, simplemente la llamamos pasándole dos datos numéricos:

```
suma(3, 2.5);
```

> **Ejercicio 1:**
>
> Crea una carpeta llamada **ejercicios2** en tu carpeta de documentos de Apache, y dentro guarda los ejercicios de esta sesión. Crea una página llamada **saludo_funciones.php**. Crea en ella una función llamada `saludo($n)`, y pon dentro el código para que saque un saludo con el nombre que se le pase como parámetro. Por ejemplo, "Hola, Nacho". Después, llama a esta función desde el código PHP para que muestre un saludo en la página con el nombre que quieras.

> **Ejercicio 2:**
>
> Crea una página llamada **contador_funciones.php** en la carpeta de ejercicios de la sesión. Crea una función llamada `cuenta($a, $b)` que reciba dos parámetros y vaya contando de un número al otro, separando los números por comas. Después, pruébala en el código PHP haciendo que cuente del 10 al 20.

2.1. Uso de parámetros por referencia

Hemos visto que, entre los paréntesis de las funciones, podemos incluir unos datos llamados argumentos o parámetros. Estos parámetros nos sirven para darle información a la función para poder realizar sus operaciones, como en el ejemplo anterior, donde le pasamos los dos números que tiene que sumar.

En ocasiones nos puede interesar que alguno de esos parámetros sea modificable, es decir, que la función pueda modificar directamente el valor del parámetro. En este caso, debemos pasar el parámetro por referencia, y eso en PHP se consigue anteponiéndole el símbolo & al parámetro. Por ejemplo, la siguiente función calcula la suma de los parámetros $n1 y $n2 y guarda el resultado en el parámetro $resultado, que se pasa por referencia.

```php
function sumaNumeros($n1, $n2, &$resultado)
{
    $resultado = $n1 + $n2;
}
```

A la hora de utilizar esta función, tendremos que pasarle tres parámetros, siendo el tercero de ellos una variable que poder modificar y guardar el resultado de la suma:

```php
$result = 0;
sumaNumeros(3, 30, $result);
// En este punto, la variable $result vale 33
```

> **Ejercicio 3:**
>
> Crea una página llamada **intercambia.php** en la carpeta de ejercicios. Añade dentro una función llamada `intercambia` que reciba 2 parámetros numéricos por referencia, y lo que haga sea intercambiar sus valores. Es decir, si recibe el parámetro `$a` y el `$b`, debe hacer que `$a` tome el valor de `$b`, y `$b` tome el valor de `$a`.

2.2. Uso de parámetros por defecto

También podemos incluir en la función parámetros con valores por defecto, de forma que, si no los ponemos, automáticamente toman el valor por defecto que hayamos indicado.

Estos parámetros deben ponerse todos al final, tras los parámetros "obligatorios", y debemos tener cuidado cuando no rellenamos alguno de ellos, para seguir el orden en que están. Por ejemplo, la siguiente función tiene los parámetros del nombre del alumno, su teléfono, una nota del examen y un año de nacimiento. La nota del examen tiene el valor por defecto de 0, y el año de nacimiento, por defecto es 1995.

```php
function datosAlumno($nombre, $telefono, $nota = 0, $aNacimiento = 1995)
{
    ...
}
```

Si utilizamos esta función, tenemos varias alternativas: la primera es utilizarla rellenando todos los parámetros:

```php
datosAlumno("Nacho Iborra", "611223344", 8, 1978);
```

La segunda es utilizarla dejando vacío algún parámetro que tenga valor por defecto. En este caso, si no ponemos la nota pero sí el año, entonces el año se asignaría a la nota, porque está en tercera posición en los

paréntesis:

```
datosAlumno("Nacho Iborra", "611223344", 1978);
// En este caso, la nota sería 1978, y el año de nacimiento 1995
```

En cambio, si rellenamos la nota, y dejamos vacío el año, entonces el año toma su valor por defecto.

```
datosAlumno("Nacho Iborra", "611223344", 8);
// En este caso, la nota sería 8 y el año de nacimiento 1995
```

En definitiva, si dejamos vacío algún parámetro que no esté al final, los de la derecha ocupan su lugar y se asignan a datos equivocados. En estos casos es mejor dejar vacíos todos, o rellenarlos con un valor por defecto para no descuadrar los demás.

3. Retorno de valores

Una función, además de hacer operaciones, y poder modificar parámetros, puede devolver un resultado de una operación (que puede ser matemática, una comprobación, etc.). Para hacer que una función devuelva un resultado, se utiliza la palabra `return` seguida del dato a devolver. Por ejemplo, la siguiente función devuelve la suma de los dos números que recibe como parámetros.

```
function sumaNumeros($a, $b)
{
    return ($a+$b);
}
```

Después, desde el código PHP, normalmente asignaremos lo que devuelve la función a alguna variable o expresión:

```
$resultado = sumaNumeros(3, 10);
// Aquí, la variable $resultado valdría 13
```

> **Ejercicio 4:**
>
> Crea una página llamada **descuento.php** en la carpeta de ejercicios. Crea una función llamada `calculaDescuento` que reciba un parámetro `$precio` con el precio de una compra, y un parámetro opcional llamado `$descuento` con el porcentaje de descuento a aplicar. Si no se pone este segundo parámetro, el valor por defecto será 0. La función devolverá con un *return* el precio con el

> descuento aplicado. Utiliza después la función desde el código PHP para calcular el descuento de un precio de 250 euros con un 10% de descuento, y el de un precio de 85 euros sin indicar descuento.

3.1. *Type hinting*

A pesar de que PHP es un lenguaje débilmente tipado (es decir, no tenemos que especificar de qué tipo de dato son las variables y parámetros que utilizamos), sí que permite indicar el tipo de dato en los parámetros y tipo de retorno de las funciones, si lo queremos, para "obligar" a que los datos que se pasen y recojan sean de esos tipos.

```
// Comprueba si dos números son iguales
function iguales(int $param1, int $param2): boolean
{
    return $param1 === $param2;
}
```

Puedes obtener más información al respecto aquí.

3.2. Funciones anónimas

Igual que ocurre con otros lenguajes, como JavaScript, podemos definir funciones anónimas. Es decir, funciones sin nombre, que pueden asignarse a una variable, o utilizarse en un punto determinado del programa para invocarlas directamente.

El siguiente ejemplo utiliza la función *array_filter* de PHP, que hace un filtrado de los elementos de un array. Como segundo parámetro, usamos una función anónima que especifica el criterio de filtrado para cada dato `$n` del array: nos quedamos con los números que sean múltiplos de 5:

```
$numeros = [12, 18, 5, 11, 10, 95, 3];

$multiplos5 = array_filter($numeros, function($n) {
    return $n % 5 == 0;
});

// $multiplos5 = [5, 10, 95]
```

4. Modularizando el código

La definición de funciones en PHP es una herramienta poderosa, como en cualquier otro lenguaje, porque nos permite reutilizar un mismo código en varios puntos de la aplicación, invocando varias veces la misma función. Pero... ¿qué ocurriría si queremos invocar una misma función en diferentes páginas PHP de nuestra

aplicación? Con lo que hemos visto hasta ahora, tendríamos que volver a definir la misma función en cada página. Afortunadamente, esto no tiene que ser así.

PHP ofrece un mecanismo de inclusión de contenidos en otras páginas PHP a través de cuatro directivas:

- **include fichero**: incluye el contenido del fichero indicado en el actual, en el punto donde se invoca esta directiva. Es básicamente como un *copiar-pegar* del fichero indicado en el punto de inclusión, pero sin repetir el código de nuevo.
- **include_once fichero**: similar al anterior, pero sólo lo incluye si no ha sido incluido ya anteriormente en el fichero (para evitar repetir inclusiones)
- **require fichero**: similar a include, pero emite un error fatal si no se encuentra el fichero indicado (mientras que *include* sólo emite un *warning*, pero sigue cargando la página). Todo dependerá de la importancia del fichero que estemos incluyendo.
- **require_once fichero**: similar al anterior, pero sólo carga el fichero si no ha sido incluido anteriormente.

Podemos utilizar estas directivas para varios fines:

- Definir un mismo contenido común a varias páginas (por ejemplo, mismo encabezado y/o mismo pie de página) e incluirlas en las distintas páginas que lo compartan. Aquí podemos elegir si utilizar *include* o *require*, dependiendo de lo vital que sea el contenido que estemos cargando para el correcto funcionamiento de la página.
- Definir un conjunto de funciones a compartir por varias partes de la aplicación, e incluirlos en las páginas que los necesiten (en este caso, lo normal sería utilizar *require* o *require_once*, ya que de lo contrario intentaríamos usar funciones que no existirían).

Los ficheros incluidos tienen a menudo una extensión especial, que puede ser **.inc** o bien **.inc.php**, para saber que no son fichero autónomos, sino que necesitan ser incluidos en otros.

Por ejemplo, podríamos definir un fichero llamado **funciones.inc** con algunas funciones de uso común:

```
<?php

function saluda()
{
    echo "Hola, buenas";
}

function suma($a, $b)
{
    echo a+b;
}
?>
```

Y después, utilizar este fichero en las páginas que lo necesiten, de este modo:

```
<?php
include "funciones.inc";
?>
<!DOCTYPE html>
...
<?php
suma(3, 5);
?>
...
```

> **Ejercicio 5:**
>
> Crea un archivo llamado **funciones.inc** en la carpeta de ejercicios de la sesión y añade dentro las funciones que hemos definido en los ejercicios anteriores. Incluye el fichero en cada uno de los ejercicios para que pueda hacer uso de las funciones que necesite.

5. Algunas funciones útiles predefinidas en PHP

PHP dispone de multitud de funciones para diferentes propósitos: manejo de textos, conexión con bases de datos, uso de expresiones regulares, etc. Veremos aquí algunos de estos conjuntos de funciones (salvo el de acceso a base de datos, que lo veremos con más detalle más adelante).

5.1. Funciones para manejo de textos

Veamos algunas funciones predefinidas que pueden sernos útiles para manejar datos de texto, como por ejemplo los que nos pueda enviar el usuario desde un formulario:

- La función `trim(cadena)` elimina espacios del principio y del final de la cadena, incluyendo tabulaciones o saltos de línea. Es especialmente útil para borrar espacios que se han quedado puestos accidentalmente.

```
$texto = "   Esto es un texto con espacios   ";
$texto2 = trim($texto);
// En este punto, $texto2 vale "Esto es un texto con espacios"
```

- La función `number_format(numero, decimales)` crea un texto mostrando el número indicado, con los decimales que se digan.

```
$numero = 3.14159;
$texto = number_format($numero, 2);
// En este punto, $texto vale 3.14
```

- La función `htmlspecialchars(cadena)` formatea la cadena sustituyendo algunos símbolos especiales (como por ejemplo `&`, `"`, `<` ...) por sus correspondientes códigos HTML (`&`, `&quot;`, `<` ...)

```
$texto = "if (var1 < var2 && ...";
$textoHTML = htmlspecialchars($texto);
// En este punto, $textoHTML vale "if (var1 &lt; var2 &amp;&amp; ..."
```

- Las funciones `strtoupper(cadena)` y `strtolower(cadena)` convierten toda la cadena a mayúsculas y minúsculas, respectivamente.

```
$texto = "Hola, buenas Tardes";
$textoMayus = strtoupper($texto);
// En este punto, $textoMayus vale "HOLA, BUENAS TARDES"
$textoMinus = strtolower($texto);
// En este punto, $textoMinus vale "hola, buenas tardes"
```

- La función `explode(separador, cadena)` devuelve una lista (array) con todas las partes de la cadena en que se puede dividir cortando por el separador indicado. La función `implode(separador, array)` combina todos los elementos de la lista (array), uniéndolos con el separador indicado.

```
$texto = "uno-dos-tres-cuatro";
$partes = explode('-', $texto);
// En este punto, $partes es una lista o array. En su posición 0 está
// la palabra 'uno', en la 1 está 'dos', en la 2 está '3' y en la 3
// está 'cuatro'
$texto2 = implode(';', $partes);
// Aquí $texto2 vale 'uno;dos;tres;cuatro'
```

- La función `substr(cadena, comienzo, fin)` obtiene una subcadena de la cadena, a partir de la posición de comienzo indicada hasta (opcionalmente) la posición de fin indicada (si no se pone fin, se toma hasta el final de la cadena).

```
$texto = "Hola, buenas tardes";
$texto2 = substr($texto, 6, 11);
// Aquí $texto2 vale 'buenas'
```

- La función `strcmp(cadena1, cadena2)` compara las dos cadenas viendo cuál es mayor alfabéticamente. Dará un número positivo si la *cadena1* es mayor, negativo si la *cadena2* es mayor, o cero si son iguales. La función `strcasecmp(cadena1, cadena2)` funciona como la anterior, pero sin

distinguir mayúsculas y minúsculas (las mayúsculas, si no se indica lo contrario, se consideran menores que las minúsculas).

```
$texto1 = "hola";
$texto2 = "adiós";
$resultado = strcmp($texto1, $texto2);
// Aquí $resultado es > 0, porque "hola" es mayor que "adiós".
```

- La función `strlen(cadena)` indica cuántos caracteres tiene la cadena.

```
$texto = "Hola, buenas tardes";
$caracteres = strlen($texto);
// Aquí $caracteres valdrá 19
```

- La función `strpos(cadena, parte)` indica en qué posición está la primera ocurrencia de la cadena *parte* en la cadena *cadena* (o un número negativo si no se encuentra).

```
$texto = "Hola, buenas tardes";
$posicion = strpos($texto, "buenas");
// Aquí $posicion valdrá 6
```

- La función `str_replace(viejacadena, nuevacadena , cadena)` reemplaza todas las ocurrencias de la cadena *viejacadena* por la cadena *nuevacadena* en la cadena global *cadena*.

```
$errores = "Un téxto de prueba con acento en la palabra téxto";
$corregido = str_replace("téxto", "texto", $errores);
// Aquí corregido vale "Un texto de prueba con acento en la palabra texto"
```

Ejercicio 6:

Crea una página llamada **manejo_textos.php**. Realiza en ella las siguientes operaciones:

- Crea una variable `$radio` con un radio de circunferencia (el que quieras).
- Crea una variable `$area` y calcula en ella el área del círculo, como has hecho en algún ejercicio anterior (PI * radio, definiendo la constante PI).
- Crea una variable `$textoResultado` que diga "El área calculada del círculo es" y luego ponga la variable `$area`, mostrando sólo 2 decimales (utiliza la función `number_format`). Muestra luego esta variable por pantalla con un *echo*.
- Crea una variable `$textoResultadoMayus` que convierta el texto anterior a mayúsculas, usando la función `strtoupper`. Muestra también esta variable por pantalla.

2

- Crea una variable llamada `$textoResultadoModificado` que reemplace la palabra "calculada" por la palabra "obtenida", usando la función `str_replace`, en la variable `$textoResultado`.
- Averigua la longitud del texto de la variable anterior usando la función `strlen`.
- Averigua en qué posición del texto de la variable anterior se encuentra la palabra "círculo", usando la función `strpos`.
- Crea una variable `$numeros` que tenga el valor "1,2,3,4,5", y utiliza la función `explode` para quedarte con los números por separado. Sácalos por pantalla, separados por el signo "+" ("1+2+3+4+5"), y después, intenta sumarlos entre sí y mostrar el resultado de la suma a continuación (al final, te quedará algo como "1+2+3+4+5=15").

5.2. Funciones para manejo de fechas y horas

Existen algunas funciones que pueden sernos muy útiles para manejar fechas y horas, y poderlas procesar o almacenar correctamente en bases de datos.

- `time()`: nos da el nº de segundos que han transcurrido desde el 1/1/1970. Esto nos servirá para establecer marcas temporales (*timestamps*), o para convertir después esta marca temporal en una fecha y hora concretas.
- `checkdate(mes, dia, año)` comprueba si la fecha formada por el mes, día y año que recibe como parámetros es correcta o no (da un valor de verdadero o falso)
- `date(formato, fecha)` obtiene una cadena de texto formateando la fecha con el formato indicado. Si no se indica ninguna fecha, se le aplicará el formato indicado a la fecha actual. Dentro del formato, podemos usar diferentes patrones, dependiendo del tipo de formato que queramos. Usaremos los símbolos d/D (para día numérico o con letra), m/M (para mes numérico o abreviado), y/Y (año de dos o cuatro dígitos), h/H (hora de 12 o 24 horas), i (minutos), s(segundos), y otras variantes que se pueden consultar en la documentación oficial

```
$fechaActual = time();
$textoFecha = date("d/m/Y H:i:s");
// Suponiendo que $fechaActual sea, por ejemplo, el 3 de noviembre de 2014
// a las 18:23:55, entonces $textoFecha sería '03/11/2014 18:23:55'
$esCorrecta = checkdate(15, 11, 2014);
// La variable $esCorrecta sería FALSE, porque 15 no es un mes válido
```

- `strtotime(texto)` convierte un texto que intenta representar una fecha en una fecha determinada. La fecha se representa en formato *mes/día/año* o *mes-día-año*, normalmente. Esta fecha sería incorrecta porque no existe el mes 21:

```
$fecha = strtotime("21/12/2013");
```

> **Ejercicio 7:**

> Crea una página llamada **ahora.php** en tu carpeta de ejercicios. Saca en la página la fecha y hora actuales en un formato como este: *07 Mar 2022 - 21:55:12*

5.3. Funciones para gestión de expresiones regulares

Al igual que otros lenguajes como JavaScript, desde PHP también podemos utilizar expresiones regulares para procesar ciertos textos y extraer patrones de ellos. La sintaxis de las expresiones regulares es muy similar a la de JavaScript, y se basa en un conjunto de símbolos y expresiones que podemos utilizar para representar los distintos tipos de caracteres de un texto (números, letras, espacios, etc.).

Símbolo	Significado
`^`	Se utiliza para indicar el comienzo de un texto o línea
`$`	Se utiliza para indicar el final de un texto o línea
`*`	Indica que el carácter anterior puede aparecer 0 o más veces
`+`	Indica que el carácter anterior puede aparecer 1 o más veces
`?`	Indica que el carácter anterior puede aparecer 0 o 1 vez
`.`	Representa a cualquier carácter, salvo el salto de línea
`x|y`	Indica que puede haber un elemento x o y
`{n}`	Indica que el carácter anterior debe aparecer n veces
`{m, n}`	Indica que el carácter anterior debe aparecer entre m y n veces
`[abc]`	Cualquiera de los caracteres entre corchetes. Podemos especificar rangos con un guión, como por ejemplo `[A-L]`
`[^abc]`	Cualquier carácter que no esté entre los corchetes
`\b`	Un límite de palabra (espacio, salto de línea...)
`\B`	Cualquier cosa que no sea un límite de palabra
`\d`	Un dígito (equivaldría al intervalo `[0-9]`)
`\D`	Cualquier cosa que no sea un dígito (equivaldría a `[^0-9]`)
`\n`	Salto de línea
`\s`	Cualquier carácter separador (espacio, tabulación, salto de página o de línea)
`\S`	Cualquier carácter que no sea separador
`\t`	Tabulación
`\w`	Cualquier alfanumérico incluyendo subrayado (igual a `[A-Za-z0-9_]`)
`\W`	Cualquier no alfanumérico ni subrayado (`[^A-Za-z0-9_]`)

Algunos símbolos especiales (como el punto, o el +), se representan literalmente anteponiéndoles la barra invertida. Por ejemplo, si queremos indicar el carácter del punto, se pondría `\.` Veamos algunos ejemplos:

Ejemplo	Significado
`^\d{9,11}$`	Entre 9 y 11 dígitos
`^\d{1,2}\/\d{1,2}\/\d{2,4}$`	Fecha (d/m/a)

A partir de esa sintaxis, disponemos de las siguientes funciones para procesar expresiones regulares:

- `preg_match(patron, texto)` devuelve 1 si el texto encaja en el patrón

```
$dni = "11223344A";
if (preg_match("/^\d{9}[A-Z]$/", $dni) == 1)
{
    echo "DNI válido";
}
```

- `preg_replace(patron, reemplazo, cadena)` examina la cadena buscando coincidencias del patrón, y reemplaza el texto encontrado por el texto de reemplazo.
- `preg_split(patron, cadena, limite)` devuelve un array o lista de cadenas, resultado de todas las ocurrencias del patrón en la cadena. Podemos especificar opcionalmente un límite de cadenas devueltas.

Además, podemos trabajar con expresiones regulares formando **grupos**. Esto se refiere a que, en un patrón de expresión regular podemos definir partes de esa expresión entre paréntesis, formando lo que se llaman grupos, cuando queremos que, a la vez que se encuentran coincidencias del patrón, se extraigan partes del mismo con información.

Por ejemplo, la siguiente variable tiene una fecha. Utilizamos una expresión regular para extraer el día, mes y año por separado, creando grupos para cada cosa. Como resultado, se obtiene una lista o array donde, en la primera posición, se tiene la fecha completa detectada, y en las siguientes 3 posiciones se tienen los tres grupos identificados (día, mes y año, respectivamente).

```
$fecha = "03-11-2014";
if (preg_match("/^([0-9]{2})-([0-9]{2})-([0-9]{4})$/", $fecha, $partes) == 1)
{
    echo "La fecha completa es " . $partes[0];
    echo "El día es " . $partes[1];
    echo "El mes es " . $partes[2];
    echo "El año es " . $partes[3];
} else {
    echo "Formato de fecha no válido";
}
```

> **Ejercicio 8:**
>
> Crea una página llamada **comprueba_hora.php** en tu carpeta de ejercicios. Crea una variable de texto con una hora en ella (por ejemplo, "21:30:12"), y luego procésala para extraer por separado la hora, el minuto y el segundo, y comprobar si es una hora válida. Por ejemplo, la hora anterior sí debería ser válida, pero si ponemos "12:63:11" no debería serlo, porque 63 no es un minuto válido.

5.4. Funciones para manejo de ficheros

A veces nos puede resultar útil leer un fichero de texto o escribir información en él. Existen multitud de funciones en PHP para abrir un fichero, leerlo línea a línea, o leer o guardar un conjunto de bytes... Vamos a ver aquí sólo algunas de las funciones más útiles para manejo de ficheros.

- `readfile(fichero)` lee un fichero entero y lo vuelca al buffer de salida (es decir, a la página que se está generando, si estamos en una página PHP).
- `file(fichero)` lee un fichero entero y devuelve un array o lista, donde en cada posición hay una línea del fichero
- `file_get_contents(fichero)` lee un fichero entero y lo devuelve en una cadena (no en un array, como la anterior)
- `file_put_contents(fichero, texto)` escribe la cadena de texto en el fichero indicado, sobreescribiendo su anterior contenido si lo tenía. En el caso de que no queramos sobreescribir, sino añadir, pondremos un tercer parámetro con la constante FILE_APPEND.
- `file_exists(fichero)` devuelve TRUE o FALSE dependiendo de si el fichero indicado existe o no
- `filesize(fichero)` devuelve el tamaño en bytes del fichero, o FALSE si no existe

El siguiente ejemplo lee un archivo llamado *libro.txt* (en la misma carpeta que el archivo actual) añade la palabra "Fin" al final y vuelve a guardar su contenido.

```
$fichero = "libro.txt";
if (file_exists($fichero))
{
    $contenido = file_get_contents($fichero);
    $contenido .= "Fin";
    file_put_contents($fichero, $contenido);
}
```

Este mismo ejemplo lo podríamos haber hecho así también, añadiendo la nueva palabra Fin al final de lo existente:

```php
$fichero = "libro.txt";
if (file_exists($fichero))
{
      file_put_contents($fichero, "Fin", FILE_APPEND);
}
```

Ejercicio 9:

Crea una página llamada **copia_seguridad.php**. En la misma carpeta, crea un archivo llamado *datos.txt* con tus datos personales en varias líneas (Nombre, Dirección, Teléfono y E-mail, tuyos o inventados). Haz que la página php lea el archivo y lo guarde en otro llamado *copia_datos.txt*, en la misma carpeta.

Arrays, errores y otros conceptos

En este documento veremos algunos aspectos adicionales y básicos del uso de PHP, como la gestión de arrays y errores.

1. Uso de arrays en PHP

Las tablas o arrays nos permiten almacenar varios datos en una sola variable, de forma que podemos acceder a esos datos utilizando distintos tipos de índices. En PHP existen tres tipos de arrays:

- **Numéricos**: la posición que ocupa cada elemento en el array la indica un número, y podemos acceder a esa posición indicando ese número entre corchetes. Las posiciones empiezan a numerarse desde la 0.
- **Asociativos**: la posición que ocupa cada elemento en la lista viene dada por un nombre, y podemos localizar cada elemento a través del nombre que le hemos dado, llamado *clave*
- **Mixtos**: son arrays de varias dimensiones, donde en algunas se utilizan índices numéricos y en otras índices asociativos.

1.1. Arrays numéricos

Estos arrays podemos crearlos de tres formas posibles:

- Indicando entre paréntesis sus elementos, y anteponiendo la palabra `array`

```
$tabla = array('Uno', 'Dos', 'Tres');
```

- Indicando a mano el índice que queremos rellenar, y el valor que va a tener (los índices intermedios que queden sin valor se quedarán como huecos vacíos)

```
$tabla[0] = 'Uno';
$tabla[1] = 'Dos';
$tabla[2] = 'Tres';
```

- Indicando el nombre del array con corchetes vacíos cada vez que queramos añadir un elemento. Así, se añade al final de los que ya existen:

```
$tabla[] = 'Uno';
$tabla[] = 'Dos';
$tabla[] = 'Tres';
```

Después, para sacar por pantalla algún valor, o usarlo en alguna expresión, pondremos el nombre del array y, entre corchetes, la posición que queremos:

```
echo $tabla[1];     // Sacaría 'Dos' en los casos anteriores
```

1.2. Arrays asociativos

En este caso, al crear el array debemos indicar, además de cada valor, la clave que le vamos a asociar, y por la que lo podemos encontrar, separados por el símbolo "=>". Por ejemplo, este array guarda para cada nombre de alumno su nota:

```
$notas = array('Manuel García'=>8.5, 'Ana López'=>7, 'Juan Solís'=>9);
```

También podemos rellenarlo, como en el caso anterior, indicando entre corchetes cada clave, y luego su valor:

```
$notas['Manuel García'] = 8.5;
$notas['Ana López'] = 7;
...
```

Después, si queremos sacar la nota del alumno Manuel García, por ejemplo, pondremos algo como:

```
echo $notas['Manuel García'];
```

En este tipo de arrays no podremos usar índices numéricos, porque no hay posiciones numéricas.

1.3. Arrays multidimensionales

Los arrays creados anteriormente son unidimensionales (sólo hay una lista o fila de elementos). Pero podemos tener tantas dimensiones como queramos. Es habitual encontrarnos con arrays bidimensionales (tablas), para almacenar información. En este caso, tendremos un corchete para cada dimensión. Por ejemplo, para crear una tabla como la siguiente...

34,1	141
36,4	150
33,5	155

... necesitaremos un código como este:

```
$tabla[0][0] = 34.1;
$tabla[0][1] = 141;
$tabla[1][0] = 36.4;
$tabla[1][1] = 150;
$tabla[2][0] = 33.5;
$tabla[2][1] = 155;
```

También podemos tener arrays multidimensionales de tipo asociativo, o array mixtos (donde algunas dimensiones son numéricas y otras asociativas). Por ejemplo:

```
$tabla2 =
array(
    array('nombre' => 'Juan García',
        'dni'=> '11111111A',
        'idiomas' => array('inglés', 'valenciano', 'español')
    ),
    array('nombre' => 'Elisa Rodríguez',
        'dni' => '22222222B',
        'idiomas' => array('francés', 'español')
    )
);
```

Podríamos mostrar el segundo idioma hablado por la segunda persona con:

```
echo $tabla2[1]['idiomas'][1];
```

Podemos crear igualmente estos arrays usando corchetes, definiendo lo que queremos en cada dimensión:

```
$tabla2[0]['nombre'] = 'Juan García';
$tabla2[0]['dni'] = '11111111A';
$tabla2[0]['idiomas'][0] = 'inglés';
$tabla2[0]['idiomas'][1] = 'valenciano';
...
```

1.4. Recorrido de arrays

Para recorrer arrays unidimensionales podemos utilizar la expresión `foreach`, que nos devuelve el valor de cada posición, independiente de si es un array numérico o asociativo:

```
foreach ($notas as $nota)
{
    echo $nota;
}
```

También podemos usar una estructura `for` que cuente hasta el tamaño del array (función *count*):

```
for($i = 0; $i < count($notas); $i++)
{
    echo $notas[$i];
}
```

Para arrays asociativos, también podemos usar esta instrucción *foreach*, indicando en la parte derecha del *as* dos variables (una para la clave y otra para el valor):

```
foreach ($notas as $alumno=>$nota)
{
    echo "El alumno $alumno tiene un $nota";
}
```

En el caso de arrays bidimensionales numéricos, para recorrer todos los elementos de un array como el de la tabla numérica anterior, necesitaremos un doble bucle (también llamado bucle anidado): uno que recorra las filas, y otro las columnas:

```php
// Filas
for ($i = 0; $i < count($tabla); $i++)
{
    // Columnas
    for ($j = 0; $j < count($tabla[0]); $j++)
    {
        echo $tabla[$i][$j];
    }
}
```

1.5. Funciones para arrays

PHP dispone de varias funciones útiles a la hora de manipular arrays. Algunas de las más habituales son:

- **count(array)** nos indica cuántos elementos tiene el array. Es útil para utilizarlo en bucles y saber cuántas repeticiones podemos hacer sobre el array.
- **sort(array)** y **rsort(array)** ordenan y reindexan un array numérico (la segunda en orden decreciente)
- **asort(array)** y **arsort(array)** ordenan y reindexan un array asociativo (la segunda en orden decreciente), por sus valores.
- **ksort(array)** y **krsort(array)** ordenan un array asociativo por sus claves (la segunda en orden decreciente).
- **usort(array, funcion)** ordena un array según la función que defina el usuario como segundo parámetro.
- **array_filter(array, funcion_filtrado)** devuelve un array con los elementos del array original (pasado como parámetro) que pasan la función de filtrado indicada.
- Podemos, además, usar la función **print_r** para sacar la información del array de forma legible

Veamos algunos ejemplos:

```php
$numeros = [2, 5, 3, 9, 6];
// Ordenamos un array de enteros de mayor a menor
rsort($numeros);  // [9, 6, 5, 3, 2]

$notas = array('Manuel García'=>8.5, 'Ana López'=>7, 'Juan Solís'=>9);
// Ordenamos un array de alumnos y notas de menor a mayor nota
asort($notas);  // Ana López = 7, Manuel García = 8.5, Juan Solís = 9
print_r($notas);
// Ordenamos un array de alumnos por nombre de menor a mayor
ksort($notas);  // Ana López = 7, Juan Solís = 9, Manuel García = 8.5
print_r($notas);

$numeros = [12, 18, 5, 11, 10, 95, 3];
// Filtramos los múltiplos de 5 del array
$multiplos5 = array_filter($numeros, function($n) {
    return $n % 5 == 0;
});
// $multiplos5 = [5, 10, 95]
print_r($multiplos5);

$alimentos = array(
    array("nombre" => "Arroz", "precio" => 1.95),
    array("nombre" => "Carne picada", "precio" => 3.45),
    array("nombre" => "Tomate frito", "precio" => 2.15)
);
// Ordenamos array por precio de menor a mayor
usort($alimentos, function($item1, $item2) {
    return $item1["precio"] <=> $item2["precio"];
});
print_r($alimentos);
```

Ejercicio 1:

Crea una carpeta llamada **ejercicios3** donde colocar los ejercicios de esta sección. Define dentro una página llamada **tabla_multiplicar.php** en tu carpeta de ejercicios. Crea en ella un array numérico bidimensional donde en cada fila almacenes la tabla de multiplicar de un número del 0 al 9; debería quedarte un array como éste (sin sacar nada por pantalla):

0	0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9	10
0	2	4	6	8	10	12	14	16	18	20
...										

Después, recorre la tabla mostrando en la página la tabla de multiplicar de cada número, de forma que te quede algo así:

```
Tabla del 0:
0 x 1 = 0
0 x 2 = 0
...
Tabla del 1:
1 x 0 = 0
...
```

> **Ejercicio 2:**
>
> Crea una página llamada **coches.php**. Define dentro un array bidimensional mixto donde:
>
> - La primera dimensión sea asociativa. Aquí pondremos matrículas de coches
> - La segunda dimensión será numérica. En cada casilla guardaremos la marca, modelo y número de puertas del coche en cuestión. Por ejemplo, el coche con matrícula "111BCD" puede ser un "Ford" (casilla 0), modelo "Focus" (casilla 1) de 5 puertas (casilla 2).
>
> Rellena el array con al menos 3 o 4 coches, y después utiliza las estructuras adecuadas para recorrerlo mostrando los datos de los coches ordenados por matrícula.

2. Gestión de errores

En ocasiones podemos hacer operaciones o utilizar funciones que pueden provocar un error grave en la aplicación. Por ejemplo, una división por cero, o una lectura de un fichero que no existe. Si no controlamos esos errores, se puede "disparar" un mensaje de error en el programa que muestre su mal funcionamiento, o lo que es peor, que revele algún dato privado, como la ubicación de un fichero en el servidor, o algún nombre de usuario o contraseña. Para evitar que ciertas operaciones que puedan causar errores alteren de esa forma el funcionamiento de la aplicación web, existen varias alternativas.

2.1. Uso del operador @

El operador `@` se pone delante de una operación que puede provocar error, de manera que, si lo provoca, el programa no diga nada, no se muestre ningún mensaje. Por ejemplo, si vamos a dividir dos variables sin tener en cuenta que el divisor pueda ser cero, podríamos ponerlo así:

```
$división = @($num1/$num2);
```

2.2. Modificación del archivo php.ini

Para no mostrar los mensajes de error por pantalla, podemos editar el archivo de configuración de PHP (php.ini), y modificar el parámetro de configuración `display_errors`. Deberemos ponerlo *false*, *off* o "0" (dependiendo de la versión de PHP que tengamos), para desactivar los mensajes de error. Pero deberemos

tener en cuenta que, si hacemos esto, se desactivarán para todas las aplicaciones PHP que tengamos en el servidor, por lo que no suele ser una práctica demasiado recomendable.

2.3. Uso de la instrucción *die*

Una tercera forma de controlar los errores que se producen en una página PHP es utilizar una instrucción `die` o `exit` en el caso de que se produzca un error en una instrucción. Por ejemplo, el siguiente código intenta ver si un fichero existe, y si no, muestra el mensaje de error indicado en la instrucción `die` / `exit`:

```
file_exists("fichero.txt") or die ("No se encuentra el fichero");
```

Además, con la función `die` o `exit` deja de ejecutarse el resto de la página, con lo que termina ahí su carga. Se utiliza para instrucciones que sean de vital importancia para el funcionamiento del resto del código, y normalmente al principio de la página para no dejarla con la mitad de contenido. De lo contrario, es mejor utilizar el operador `@` para ocultar errores menos importantes.

2.4. Uso de excepciones

Las excepciones son un mecanismo que podemos emplear para, en el momento en que se produzca un error en alguna parte del código, se capture automáticamente y se pueda tratar en un bloque de código aparte.

Para trabajar con excepciones, debemos colocar todo el código que puede provocar error en un bloque llamado `try`. Si algo falla, automáticamente se salta a un bloque contiguo llamado `catch`, donde podremos tratar el error y sacar el mensaje oportuno. El siguiente ejemplo trata de abrir un fichero, leerlo y guardar el contenido en un array.

```
try
{
    $contenido = file("fich1.txt");
} catch (Exception $e) {
    echo 'Se ha producido un error: ' . $e->getMessage();
}
```

Si se produce cualquier tipo de error (por ejemplo, que no exista el fichero de lectura, o no tengamos permisos para acceder a él), se provoca un error y se va directamente al bloque *catch*, mostrando el mensaje de error. Podemos utilizar la variable de tipo `Exception` que tiene el *catch* como parámetro, y su método `getMessage()` para mostrar el error concreto que se ha producido.

También podemos usar la instrucción `throw new Exception($mensaje)` para provocar una excepción en el caso de que alguna comprobación que hagamos nos dé un resultado incorrecto. Así provocamos un salto al *catch*, o un mensaje de error en la web si no lo hacemos dentro de un try. Se utiliza también para algunas funciones que no provocan excepciones por sí mismas (como por ejemplo, *file_get_contents*), para provocar el error nosotros de antemano con alguna comprobación previa.

```
try
{
    if (!file_exists("fich1.txt"))
    {
        throw new Exception("El fichero de entrada no existe");
    }
    $contenido = file_get_contents("fich1.txt");
    file_put_contents("fich2.txt", $contenido);
} catch (Exception $e) {
    echo 'Se ha producido un error: ' . $e->getMessage();
}
```

Ejercicio 3:

Haz tres copias del *Ejercicio 9* de este documento, llamadas **copia_seguridad_arroba.php**, **copia_seguridad_die.php** y **copia_seguridad_excepciones.php**. En todas ellas, cambia el nombre del fichero *datos.txt* por uno que no exista en la carpeta (por ejemplo, *aa.txt*), y controla el error que puede producirse en cada caso usando una arroba, la instrucción `die` o las excepciones, para ver cómo se controla en cada caso el error y qué hace la página.

3. Más sobre variables

Existen algunas cuestiones algo más avanzadas sobre el uso de variables en PHP que debemos conocer.

3.1. Variables globales y locales

Veremos más adelante que existen unas variables predefinidas en PHP para ciertas tareas. Por ejemplo, la variable $_REQUEST nos permitirá acceder a los datos que el usuario nos envía desde un formulario.

Otras variables que podamos crear nosotros, tendrán su ámbito según la zona donde las creemos. Por ejemplo, si creamos una variable dentro de una función, esa variable no existirá fuera de la misma, y no podremos utilizarla. Sin embargo, si creamos una variable fuera de una función, e intentamos acceder a ella dentro, podría parecer que sí es una variable global o externa a la función, pero no es así. Veamos este ejemplo:

```
$numero = 10;
...
function incrementaNumero()
{
    $numero = $numero + 1;
    echo $numero;
}
...
incrementaNumero();
```

Si ejecutamos un código como este, el comando `echo $numero` nos dirá que *numero* vale 1, cuando todo parece indicar que debería valer 11. La razón es que la variable externa `$numero` no es la misma que la variable interna `$numero` de la función. Esta última, al no tener un valor inicial asignado, empieza por 0, y al incrementarse vale 1, pero la variable `$numero` externa sigue valiendo 10, nadie la ha modificado.

Si queremos poder acceder a una variable externa a una función desde dentro de una función, deberemos definir en la función que esa variable es global, de la siguiente forma:

```
$numero = 10;
...
function incrementaNumero()
{
    global $numero;
    $numero = $numero + 1;
    echo $numero;
}
...
incrementaNumero();
```

Ahora, si ejecutamos el código, sí obtendremos lo esperado: que `$numero` vale 11.

3.2. Closures

Los *closures* son un mecanismo vinculado a las funciones anónimas, mediante el cual éstas pueden utilizar elementos externos, tales como variables. Para hacer eso, se emplea la cláusula `use` seguida de la variable (o variables) que se quieren emplear.

```
$numero = 10;
$incrementaNumero = function () use ($numero)
{
    $numero = $numero + 1;
    echo $numero;
};
$incrementaNumero();    // 11
```

3.3. Variables variables

Una funcionalidad muy característica de PHP y que no todos los lenguajes tienen es la posibilidad de definir variables cuyo nombre es a su vez variable. Así, una variable puede tomar su nombre según el valor de otra variable. Por ejemplo:

```
$var1 = "uno";
$$var1 = "otro uno";
```

La variable `$var1` es una variable normal, y guarda el valor "uno". Sin embargo, la variable `$$var1` tomará su nombre según lo que valga `$var1` (en este caso, se llamaría `$uno`). Para utilizar estas variables en instrucciones tipo *echo*, dentro de comillas dobles, se pone el nombre dinámico (empezando por el segundo dólar) entre llaves:

```
echo "La segunda variable vale ${$var1}";
```

¿Qué utilidades puede tener esto? Puede parecer que es un mecanismo un tanto enrevesado sin demasiada utilidad pero, por ejemplo, puede ser bastante útil para hacer páginas multi-idioma. Así, nos guardaremos en diferentes variables el texto a mostrar en cada idioma, y haremos que se imprima una de ellas en función de algún parámetro adicional o variable.

```
<?php
    $texto_es = "Bienvenido";
    $texto_en = "Welcome";
    $idioma = "es";
    $texto = "texto_" . $idioma;
    echo $$texto;
?>
```

> **Ejercicio 4:**

> Crea una página en la carpeta de ejercicios llamada **curriculum.php** donde, utilizando variables variables, muestres parte de tu currículum (por ejemplo, un párrafo con tus estudios y otro con los idiomas que hablas), tanto en español como en otro idioma que elijas.

Programación orientada a objetos con PHP

La Programación Orientada a Objetos es una forma de programar relativamente reciente (en PHP existe desde su versión 5), si la comparamos con la forma convencional de programar que hemos visto hasta ahora en los apartados previos, llamada programación estructurada y modular, que utiliza simplemente estructuras de control de flujo (*if*, *while*, *for*) y funciones para estructurar y dividir el código de un programa.

Mediante la Programación Orientada a Objetos, se pretende identificar cada uno de los tipos de objeto que componen un programa. A cada uno de esos tipos se le llama **clase**, y está compuesto por una serie de propiedades o **atributos**, y una serie de operaciones que puede realizar (**métodos**). Cada variable que creemos de ese tipo será un **objeto** de esa clase, con sus propios valores para sus propiedades o atributos.

1. Ejemplo sencillo

Veámoslo con un ejemplo. Imaginemos que tenemos una base de datos donde queremos almacenar el software que tenemos en casa. De cada software queremos almacenar un código identificativo, su título y su versión. También queremos que, para cualquier software, podamos mostrar sus datos por pantalla.

Para representar cualquier software que queramos gestionar, podemos crearnos una clase llamada `Software`, y definirle los atributos que va a tener (en este caso, su código, título y versión) y las operaciones que va a poder realizar, que definiremos en forma de funciones (por ejemplo, mostrar sus datos). Para hacer esto en PHP, usaremos la palabra `class` con el nombre de la clase que queramos (en este caso, *Software*), y dentro definimos sus atributos (con algún valor predeterminado, si queremos) y sus funciones o métodos.

```php
class Software
{
    private $codigo = 0;
    private $titulo = "";
    private $version = "1";

    public function MostrarDatos()
    {
        echo "<p>$this->titulo ($this->version)</p>";
    }
}
```

Una diferencia importante entre los atributos de la clase y las variables normales es que, para poderlos referenciar en cualquier método de la clase, tenemos que anteponerle el objeto `$this`, seguido del operador flecha `->` y el nombre del atributo (sin el dólar). Esto es así para evitar problemas en el caso de que alguna otra variable en la función se llame igual que el atributo.

A diferencia del resto de código PHP, no podemos interrumpir la definición de una clase para intercalar código HTML en medio, toda la clase debe estar comprendida entre el mismo par de etiquetas `<?php` y `?>`. Las palabras `private` y `public` las explicaremos más adelante.

2. Creación de objetos

Ya tenemos la clase creada. ¿Cómo usamos variables de tipo *Software*? Para poder crear variables u objetos de cualquier clase, necesitamos definir una función especial llamada **constructor**. Estas funciones pueden recibir una serie de parámetros, que normalmente son los valores que queremos asignarles a los distintos atributos. Así, por ejemplo, para nuestra clase anterior, podemos definir un constructor que reciba tres parámetros (uno para el código, otro para el título y otro para la versión) y los asigne a los correspondientes atributos:

```
class Software
{
    private $codigo = 0;
    private $titulo = "";
    private $version = "1";

    public function __construct($c, $t, $v)
    {
        $this->codigo = $c;
        $this->titulo = $t;
        $this->version = $v;
    }

    ...
}
```

Con esto, ya podemos crear variables de este tipo, usando el operador `new` para crear cada objeto. Por ejemplo, así crearíamos dos variables de tipo Software, `$s1` y `$s2`, con diferentes datos cada una (desde fuera de la clase, si queremos):

```
$s1 = new Software(1, "LibreOffice", "6.0");
$s2 = new Software(2, "GIMP", "3.8");
```

Y para llamar a las funciones de la clase y poder, por ejemplo, mostrar los datos de cada software por pantalla, haríamos algo como esto.

```
echo "<p>Datos del primer programa:</p>";
$s1->MostrarDatos();
echo "<p>Datos del segundo programa:</p>";
$s2->MostrarDatos();
```

Observa que usamos el operador `->` para llamar a la función de un objeto. Ya hemos visto algo parecido anteriormente para acceder al error de una excepción.

3. Modificadores de acceso

Cuando hemos creado la clase *Software*, delante de los atributos o propiedades de la clase hemos puesto la palabra *private*, que es un modificador de acceso que quiere decir que no se puede acceder a esos atributos desde fuera de la clase (para preservar que su valor no se modifique sin control). En cambio, las funciones tienen el modificador *public*, para poderlas llamar desde fuera.

En general, podemos utilizar tres modificadores diferentes en las clases:

- **private**: el elemento no puede ser visto desde fuera de la clase
- **public**: el elemento puede verse y utilizarse fuera de la clase. Es el modificador por defecto si no indicamos uno nosotros
- **protected**: para herencia, el elemento es visible desde la propia clase o las clases heredadas; veremos el tema de la herencia a continuación.

3.1. Getters y setters

Hemos dicho que los atributos de una clase normalmente son privados. Esto es así para no poder acceder a ellos directamente desde fuera, y cambiar su valor erróneamente. Por ejemplo, en el caso anterior, si los atributos fueran públicos, alguien podría intentar acceder al código de un programa y darle un valor negativo:

```
$s1 = new Software(...);
...
$s1->codigo = -1;
```

Para evitar esto, se suelen dejar privados, y para acceder a su valor o modificarlo, se añaden unas funciones o métodos especiales, llamadas comúnmente **getters y setters**. Los primeros (*getters*) se llaman así porque el nombre de la función suele empezar por `__get` y sirven para obtener el valor del atributo al que representan. Los segundos (*setters*) se llaman así porque la función suele empezar por `__set` y se emplean para modificar el valor del atributo. Dentro de esta segunda función, podemos hacer alguna comprobación previa antes de cambiar el valor del atributo (por ejemplo, en el caso del código, comprobar que no tenga un valor negativo).

```php
class Software
{
    private $codigo;
    private $titulo;
    private $version;
    ...

    public function __get($nombre)
    {
        if ($nombre == 'Cod')
            return $this->codigo;
        else if ($nombre == 'Titulo')
            return $this->titulo;
        else if ($nombre == 'Version')
            return $this->version;
    }

    public function __set($nombre, $valor)
    {
        if ($nombre == 'Cod' && $valor > 0)
            $this->codigo = $valor;
        else if ($nombre == 'Titulo')
            $this->titulo = $valor;
        else if ($nombre == 'Version')
            $this->version = $valor;
    }
    ...
}
```

Observa que el método `__get` recibe como parámetro un nombre (el que queramos), y luego dentro de la función emparejamos cada nombre con el atributo al que queramos enlazarlo. Si ponemos *Cod*, lo asociaremos al atributo `$codigo`, y así sucesivamente.

Con esto, si queremos cambiar el código del objeto anterior, podríamos hacerlo con su setter correspondiente, y asegurarnos de que se cambiará a un valor correcto.

```php
$s1->Cod = -1; // Esto no hará nada
$s1->Cod = 10; // Esto sí funcionará
```

La llamada al *setter* o al *getter* no es como en el resto de funciones. Tenemos que poner el nombre del objeto, el operador `->` y un nombre (de entre los que vaya a aceptar el *getter* o *setter*), y en el caso de querer asignarle un valor, se lo asignamos como si fuera una variable normal. Ese nombre, como podemos apreciar, no tiene por qué coincidir con el nombre del atributo. Dentro de la función nos encargamos de emparejar cada nombre con el atributo, así que pueden ser diferentes, ya que la asociación la hacemos nosotros a mano.

4. Herencia

La herencia es una de las herramientas más potentes de la programación orientada a objetos. Consiste en definir una clase partiendo de otra, y suponiendo que la nueva clase es un subtipo de la anterior. Por ejemplo, si tuviéramos una clase *Animal* con una serie de atributos comunes de cualquier animal (por ejemplo, color y número de patas), podríamos definir una clase *Perro* que heredara de *Animal*, con lo que ya tendría implícitamente todo lo que tuviera *Animal* (en este caso, los atributos del color y número de patas), y además, podríamos añadir a esta nueva clase las características propias de un perro (por ejemplo, el tipo de ladrido).

Volviendo a nuestro ejemplo del *Software*, imaginemos que, entre el software que tenemos en casa, tenemos varios videojuegos, y que de ellos nos interesa guardar, además del título y la versión, la plataforma para la que están hechos (PC, consola, etc.). En este caso, como los videojuegos son un subtipo de software, podemos aplicar herencia, y crear una nueva clase `Videojuego` que herede de *Software*, así:

```
class Videojuego extends Software
{
    private $plataforma;

    public function __construct($c, $t, $v, $p)
    {
        $this->codigo = $c;
        $this->titulo = $t;
        $this->version = $v;
        $this->plataforma = $p;
    }

    ...
}
```

A la clase heredada (en este caso, *Software*), se le suele llamar clase base, superclase o clase padre. A la clase que hereda, se le llama subclase, clase derivada o clase hija.

Así, la clase *Videojuego* heredará todo lo que tenemos hecho en *Software*. Recuerda que los modificadores *private* impiden que el elemento que lo tiene sea visible desde fuera. Eso quiere decir que, ahora mismo, en *Videojuego*, no podríamos acceder directamente a los atributos `$codigo`, `$titulo` o `$version`, pero sí podemos acceder a los *getters* y *setters* de *Software* para darles valor o consultarlo. De todas formas, si quisiéramos acceder directamente a esos atributos, sin pasar por los *getters* y *setters*, tendríamos que definirlos como *protected* en la clase *Software*.

```
class Software
{
    protected $codigo;
    protected $titulo;
    protected $version;

    ...
}
```

Así, el constructor que hemos definido antes en *Videojuego* sí funcionaría (antes no, porque los atributos eran private).

4.1. El elemento parent

Hemos visto que el objeto `$this` nos sirve para referenciar a los atributos o elementos de una clase, y poderlos distinguir de otros externos que se llamen igual. Del mismo modo, podemos referenciar a los atributos o métodos de la clase padre mediante el objeto `parent`, con la sintaxis `parent::metodo(...)`. Así, por ejemplo, en el caso anterior, no sería necesario duplicar el código en el constructor de la clase hija para los atributos que asigna la clase padre. Podríamos poner:

```
public function __construct($c, $t, $v, $p)
{
    parent::__construct($c, $t, $v);
    $this->plataforma = $p;
}
```

> **Ejercicio 1:**
>
> Crea una carpeta **ejercicios7** para los ejercicios de esta sesión. Dentro crea una página llamada **clases.php** con una clase llamada **Persona** que tenga como atributos un DNI, un nombre y un email. Crea un constructor que permita rellenar esos tres atributos, y los *getters* y *setters* correspondientes. Define también un método *Mostrar* para sacar por la página los datos de la persona (en un párrafo, separados por guiones). Define adecuadamente la visibilidad (pública o privada) de cada atributo o método.
>
> Crea una segunda clase llamada **Estudiante** que herede de *Persona*, y añada un atributo llamado *numExpediente*. Crea su constructor, sus *getters* y *setters* y su correspondiente método *Mostrar*.
>
> Fuera de las clases, entre el código HTML de la página, crea un objeto de cada tipo (una *Persona* y un *Estudiante*), con los valores que quieras, llama después a algún setter de cada una para cambiar el valor de algún atributo, y finalmente llama a sus métodos *Mostrar* para que saquen la información de cada uno.