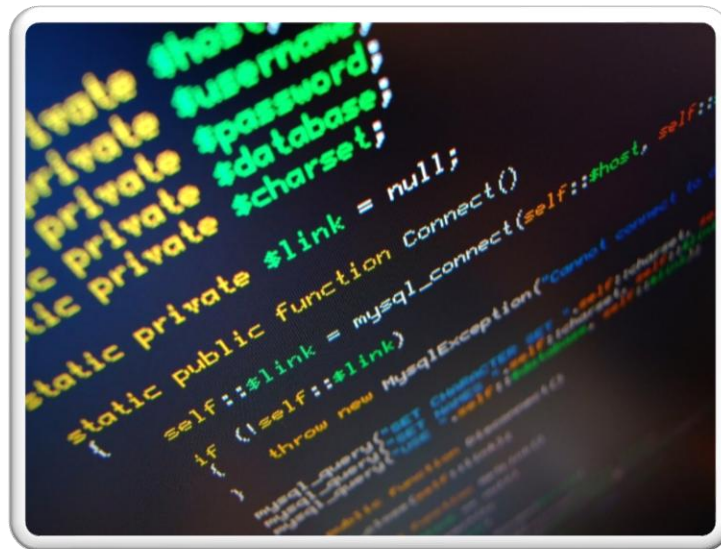
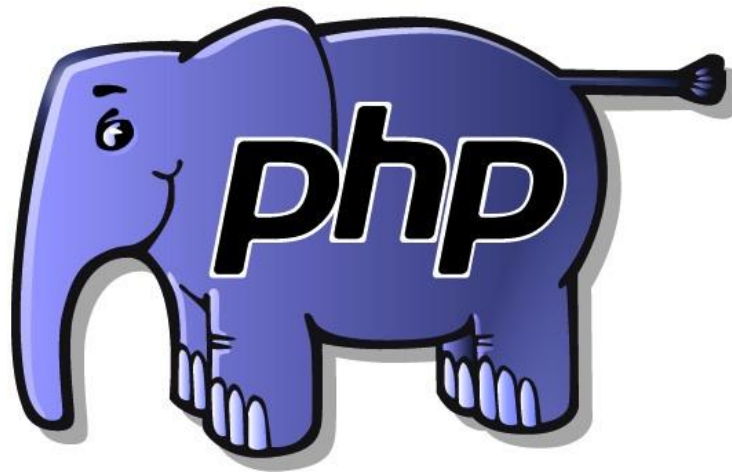


Desarrollo Web en Entorno Servidor

2º CFGS

Desarrollo de Aplicaciones Web



INDICE

1. Introducción al desarrollo web del lado servidor	6
¿Qué es la programación de sitios web de lado servidor?	6
Sitios estáticos	6
Sitios dinámicos	6
¿Son iguales la programación del lado-servidor y lado-cliente?	7
¿Qué se puede hacer en el lado-servidor?	8
Aplicaciones Web	10
Programación en entorno servidor.	10
Arquitecturas y plataformas.	11
Código embebido en lenguajes de marca	12
2. Introducción PHP	13
Visual Studio Code	14
3. Conceptos básicos: Variables, Operadores y formularios	2
3.1 Variables	3
3.2 Operadores aritméticos	5
3.3 Ejercicios para practicar	6
3.4 Recogida de datos por teclado mediante formularios	8
3.5 Ejercicios para practicar	10
3.6 Conceptos de la Unidad	11
3.7 Ejercicios para resolver	13
4. Estructuras de control	14
4.1 Condicionales	14
4.2 Ejercicios para practicar	19
4.3 Repetitivas	21
4.4 Ejercicios para practicar	24
4.5 Conceptos de la unidad	27
4.6 Ejercicios para resolver	29
5. Arrays	31
5.1 Arrays clásicos	31

5.2 Arrays asociativos	33
5.3 Arrays bidimensionales	34
5.4 Iterador <code>foreach</code>	35
5.5 Cómo recoger datos para un array mediante un formulario	35
5.6 Ejercicios para practicar	36
5.7 Conceptos de la unidad	40
5.8 Ejercicios para resolver	44
6. Funciones	47
6.1 Implementando funciones para reutilizar código	47
6.2 Creación de bibliotecas de funciones	49
6.4 Ejercicios para practicar	51
6.5 Ejercicios para resolver	52
6.6 Conceptos de la unidad	54
6.7 Funciones de texto	56
6.8 Ejercicios de Cadenas	57
6.9 Funciones de fecha	59
6.10 Ejercicios de Fechas	60
7. Sesiones y cookies	62
7.1 Sesiones	62
7.2 Cookies	65
7.3 Ejercicios para practicar	68
7.4 Ejercicios para resolver	70
7.5 Conceptos de la unidad	72
8. Ficheros	73
8.1 Subir un fichero a través de un formulario	73
8.2 Introducción al manejo de archivos con PHP.	73
8.3 Conceptos de la unidad	77
8.4 Listar ficheros ubicados en un directorio	78
8.5 Ejercicios para practicar	78
8.6 Ejercicios para resolver	81
9. Programación orientado a objetos	82

9.1 El paradigma de la Programación Orientada a Objetos	82
9.2 Encapsulamiento y ocultación	82
9.3 Implementación de clases en PHP	83
9.4 Herencia	87
9.5 Atributos y métodos de clase (<i>static</i>)	91
9.6 Almacenar objetos en sesiones	94
9.7 Ejercicios para resolver	96
9.8 Conceptos de la unidad	99
10. Acceso a bases de datos	102
10.1 Acceso a BBDD desde PHP	102
10.2 PDO (PHP Data Objects)	102
10.3 Operaciones sobre una tabla	104
10.4 Error al iniciar servicio MYSQL	106
10.5 Ejercicios para resolver	107
10.6 Conceptos de la unidad	109
11. Modelo Vista Controlador	111
11.1 El modelo	112
11.2 La vista	115
11.3 El controlador	117
11.4 Ejercicios para resolver	118
1.5 Conceptos de la unidad	120
12. Servicios web (web services)	122
12.1 ¿Qué son los servicios web?	122
12.2 Consumo de servicios REST	122
12.3 Estructura de JSON	127
12.4 ¿Qué son las cabeceras o headers HTTP?	129
12.5 Ejercicios para resolver	130
12.6 Creación de servicios web	132
12.7 Ejercicios para resolver	140

1. Introducción al desarrollo web del lado servidor

La mayoría de los grandes sitios web usan código del lado servidor para presentar, cuando se necesitan, diferentes datos, generalmente extraídos de una base de datos almacenada en un servidor y enviada al cliente para ser presentada mediante algún código (ej, HTML y JavaScript). Quizá el beneficio más significativo de la codificación de lado servidor es que te permite confeccionar el contenido del sitio web para usuarios individuales. Los sitios dinámicos pueden resaltar contenido que es más relevante basándose en las preferencias del usuario y sus hábitos. Puede hacer también que los sitios sean más fáciles de usar al almacenar las preferencias personales y la información - por ejemplo, reusando los detalles de la tarjeta de crédito guardados para agilizar los pagos siguientes. Puede incluso permitir la interacción con los usuarios fuera del sitio, enviando notificaciones y actualizaciones vía email o a través de otros canales. Todas estas capacidades permiten un mayor compromiso con los usuarios.

¿Qué es la programación de sitios web de lado servidor?

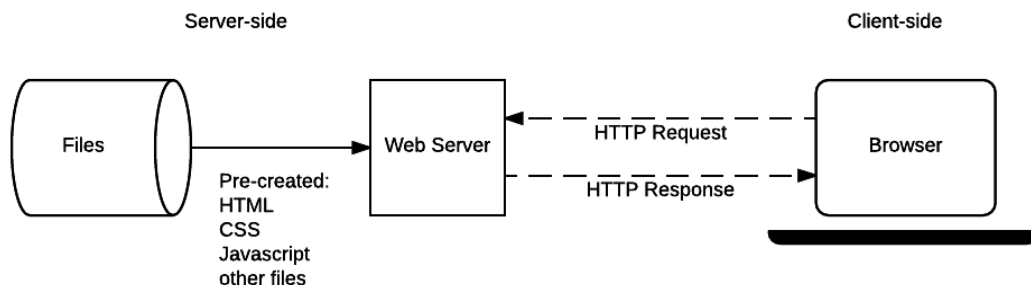
Los exploradores web se comunican con los servidores web usando el Protocolo de Transporte de Hyper Texto (HyperText Transport Protocol (HTTP)). Cuando pinchas en un enlace en una página web, envías un formulario o ejecutas una búsqueda, se envía una petición HTTP desde tu explorador web al servidor web de destino. La petición incluye un URL que identifica el recurso afectado, un método que define la acción requerida (por ejemplo, obtener, borrar o publicar el recurso), y puede incluir información adicional codificada en parámetros en el URL (los pares campo-valor enviados en una cadena de consulta (query string), como datos POST (datos enviados mediante el método POST de HTTP, HTTP POST method), o en associated cookies.

Los servidores web esperan los mensajes de petición de los clientes, los procesan cuando llegan y responden al explorador web con un mensaje de respuesta HTTP. La repuesta contiene una línea de estado indicando si la petición ha tenido éxito o no (ej, "HTTP/1.1 200 OK" en caso de éxito). El cuerpo de una respuesta exitosa a una petición podría contener el recurso solicitado (ej, una nueva página HTML, o una imagen, etc...), que el explorador web podría presentar en pantalla.

Sitios estáticos

Estas páginas se encuentran almacenadas en su forma definitiva, tal y como se crearon, y su contenido no varía. Son útiles para mostrar una información concreta, y mostrarán esa misma información cada vez que se carguen. La única forma en que pueden cambiar es si un programador la modifica y actualiza su contenido.

El diagrama de abajo muestra una arquitectura de servidor web básica correspondiente a un *sitio estático* (un sitio estático es aquél que devuelve desde el servidor el mismo contenido insertado en el código "hard coded" siempre que se solicita un recurso en particular). Cuando un usuario quiere navegar a una página, el explorador envía una petición HTTP "GET" especificando su URL. El servidor recupera de su sistema de ficheros el documento solicitado y devuelve una respuesta HTTP que contiene el documento y un estado de éxito "success status" (normalmente 200 OK). Si el fichero no puede ser recuperado por alguna razón, se devuelve un estado de error.



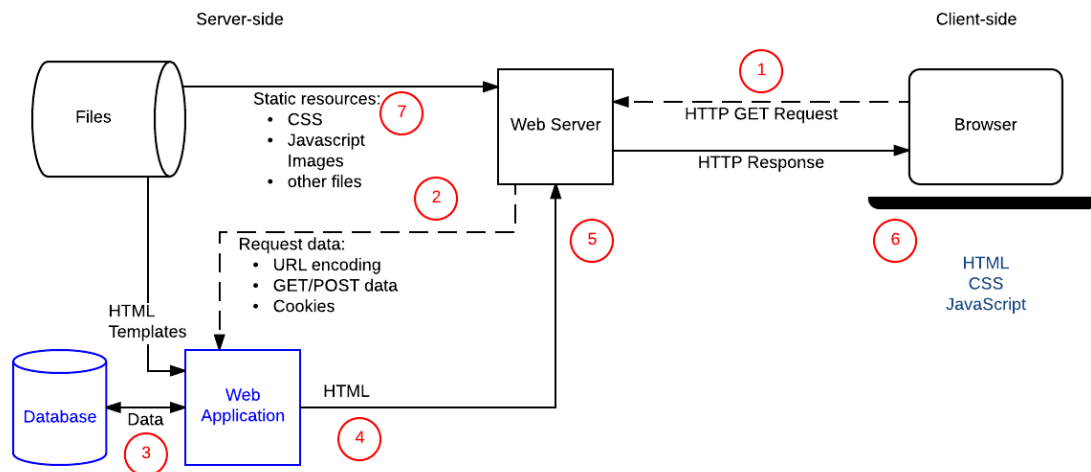
Sitios dinámicos

Dentro de las **páginas web dinámicas**, es muy importante distinguir **dos tipos**:

- Aquellas que **incluyen código que ejecuta el navegador**. En estas páginas el código ejecutable, normalmente en **lenguaje JavaScript**, se incluye dentro del HTML (o XHTML) y se descarga junto con la página. Cuando el navegador muestra la página en pantalla, ejecuta el código que la acompaña. Este código puede incorporar múltiples funcionalidades que pueden ir desde mostrar animaciones hasta cambiar totalmente la apariencia y el contenido de la página. En este módulo no vamos a ver JavaScript, salvo cuando éste se relaciona con la programación web del lado del servidor.
- Como ya sabes, hay muchas páginas en Internet que no tienen extensión .htm, .html o .xhtml. Muchas de estas páginas tienen extensiones como .php, .asp, .jsp, .cgi o .aspx. En éstas, el contenido que se descarga al navegador es similar al de una página web estática: HTML (o XHTML). Lo que cambia es la forma en que se obtiene ese contenido. Al contrario de lo que vimos hasta ahora, esas páginas no están almacenadas en el servidor; más concretamente, el contenido que se almacena no es el mismo que después se envía

al navegador. **El HTML de estas páginas se forma como resultado de la ejecución de un programa**, y esa ejecución tiene lugar en el servidor web (aunque no necesariamente por ese mismo servidor).

El diagrama de abajo muestra una arquitectura simple para un *sitio web dinámico*. Como en el diagrama previo, los exploradores web envían peticiones HTTP al servidor, el servidor procesa a continuación las peticiones y devuelve las respuestas HTTP apropiadas. Las peticiones de recursos *estáticos* son gestionadas de la misma manera que para los *sitios estáticos* (los recursos estáticos son cualquier fichero que no cambia - generalmente: CSS, JavaScript, Imágenes, ficheros PDF creados previamente, etc...)



Las peticiones de recursos dinámicos, por el contrario, ejecutan los siguientes pasos:

1. **El cliente web** (navegador) de tu ordenador **solicita** a un servidor web una **página web**.
2. **El servidor busca** esa página y la recupera.
3. En el caso de que se trate de una **página web dinámica**, es decir, que su contenido deba ejecutarse para obtener el HTML que se devolverá, **el servidor web** contacta con el módulo responsable de **ejecutar el código** y se lo envía.
4. Como parte del **proceso de ejecución**, puede ser necesario obtener información de algún repositorio, como por ejemplo consultar registros almacenados en una base de datos, imágenes, hojas de estilos, etc.
5. **El resultado** de la ejecución será una página en **formato HTML**, similar a cualquier otra página web no dinámica.
6. **El servidor web envía el resultado** obtenido al navegador, que la procesa y muestra en pantalla.

Este procedimiento tiene lugar constantemente mientras consultamos páginas web. Por ejemplo, cuando consultas tu correo en GMail, HotMail, Yahoo o cualquier otro servicio de correo vía web, lo primero que tienes que hacer es introducir tu nombre de usuario y contraseña. A continuación, lo más habitual es que el servidor te muestre una pantalla con la bandeja de entrada, en la que aparecen los mensajes recibidos en tu cuenta. Esta pantalla es un claro ejemplo de una página web dinámica. Obviamente, el navegador no envía esa misma página a todos los usuarios, sino que la **genera de forma dinámica** en función de quién sea el usuario que se conecte. Para generarla ejecuta un programa que obtiene los datos de tu usuario (tus contactos, la lista de mensajes recibidos) y con ellos compone la página web que recibes desde el servidor web.

¿Son iguales la programación del lado-servidor y lado-cliente?

Prestemos ahora nuestra atención al código involucrado en la programación de lado-servidor y lado-cliente. En cada caso, el código es significativamente diferente:

- Tienen diferentes propósitos y preocupaciones.
- Por lo general no usan los mismos lenguajes de programación (siendo la excepción el JavaScript, que puede usarse tanto en lado servidor como en lado cliente).
- Se ejecutan entornos de diferentes sistemas operativos.

El código que se ejecuta en el explorador se conoce como código de lado-cliente, y su principal preocupación es la mejora de la apariencia y el comportamiento de una página web entregada. Esto incluye la selección y estilo de los componentes UI, la creación de layouts, navegación, validación de formularios, etc. Por otro lado, la programación de sitios web de lado servidor en su mayor parte implica la elección de *qué contenido* se ha de devolver al explorador como respuesta a sus peticiones. El código de lado-servidor gestiona tareas como la validación de los datos enviados y las peticiones, usando bases de datos para almacenar y recuperar datos, y enviando los datos correctos al cliente según se requiera.

El código del lado cliente está escrito usando **HTML, CSS, y JavaScript** — es ejecutado dentro del explorador web y tiene poco o ningún acceso al sistema operativo subyacente (incluyendo un acceso limitado al sistema de ficheros).

Los desarrolladores web no pueden controlar qué explorador web usará cada usuario para visualizar un sitio web — los exploradores web proporcionan niveles de compatibilidad inconsistentes con las características de codificación lado cliente, y parte del reto de la programación de lado cliente es gestionar con dignidad las diferencias de soporte entre exploradores.

El código del lado servidor puede escribirse en cualquier número de lenguajes de programación — ejemplos de lenguajes de programación populares incluyen PHP, Python, Ruby, C# y NodeJS(Javascript). El código del lado servidor tiene acceso completo al sistema operativo del servidor y el desarrollador puede elegir qué lenguaje de programación (y qué versión específica) desea usar.

Los desarrolladores generalmente escriben su código usando web frameworks. Los web frameworks son colecciones de funciones, objetos, reglas y otras construcciones de código diseñadas para resolver problemas comunes, acelerar el desarrollo y simplificar los diferentes tipos de tareas que se han de abordar en un dominio en particular.

De nuevo, mientras que, tanto el código lado cliente y el lado servidor usan frameworks, los dominios son muy diferentes, y por lo tanto también lo son los frameworks. Los frameworks del lado cliente simplifican los diseños y las tareas de presentación mientras que los del lado servidor proporcionan un montón de funcionalidades "comunes" que tendría que haber implementado uno mismo (ej, soporte para las sesiones, soporte para los usuarios y autenticación, acceso fácil a la base de datos, librerías de plantillas, etc...).

¿Qué se puede hacer en el lado-servidor?

La programación del lado-servidor es muy útil porque nos permite distribuir *eficientemente* información a medida para usuarios individuales y por lo tanto crear una experiencia de usuario mucho mejor.

Compañías como Amazon utilizan la programación del lado-servidor para construir resultados de búsquedas de productos, hacer sugerencias sobre productos escogidos basados en las preferencias del cliente y sus hábitos de compra previos, simplificar las adquisiciones, etc. Los bancos usan la programación del lado-servidor para almacenar la información sobre las cuentas y permitir ver y realizar transacciones sólo a los usuarios autorizados. Otros servicios como Facebook, Twitter, Instagram y Wikipedia usan la programación de lado-servidor para destacar, compartir y controlar el acceso al contenido interesante.

Algunos de los usos y beneficios comunes de la programación de lado-servidor se lista debajo. Notarás que hay algo de solapamiento.

- **Almacenaje y distribución eficiente de información**

Imagina cuántos productos están disponibles en Amazon, e imagina cuántas entradas se han escrito en Facebook. Crear una página estática separada para cada producto o entrada sería completamente ineficiente.

La programación de lado-servidor nos permite por el contrario almacenar la información en una base de datos y construir dinámicamente y devolver ficheros HTML y de otros tipos (ej, PDFs, imágenes, etc.). También es posible devolver simplemente datos (**JSON, XML**, etc.) para presentar mediante las webs, frameworks adecuados del lado-cliente (esto reduce la carga de procesamiento del servidor y la cantidad de datos que se necesitan enviar).

El servidor no se limita a enviar información de las bases de datos, y podría además devolver el resultado de herramientas de software o datos de servicios de comunicación. El contenido puede incluso ser dirigido por el tipo de dispositivo cliente que lo está recibiendo.

Debido a que la información está en una base de datos, puede también ser compartida y actualizada con otros sistemas de negocio (por ejemplo, cuando se venden los productos online o en una tienda, la tienda debería actualizar su base de datos de inventario).

Nota: Tu imaginación no tiene que trabajar duro para ver el beneficio de la codificación de lado-servidor para el almacenaje y distribución de información:

1. Vete a Amazon o a cualquier otro sitio de comercio electrónico "e-commerce".
2. Busca por un número de palabras clave y nota como la estructura de la página no cambia, incluso aunque cambien los resultados.
3. Abre dos o tres productos diferentes. Fíjate de nuevo como tienen una estructura y diseño común, pero el contenido para los diferentes productos ha sido extraído de la base de datos.

Para un término de búsqueda común (digamos "pez") puedes ver literalmente millones de valores retornados. Usar una base de datos permite que éstos sean almacenados y compartidos de forma eficiente, y permite que la presentación de la información esté controlada en un solo sitio.

- **Experiencia de usuario personalizada**

Los servidores pueden almacenar y usar la información acerca de los clientes para proporcionar una experiencia de usuario conveniente y dirigida. Por ejemplo, muchos usuarios almacenan tarjetas de crédito de forma que los detalles no tienen que ser introducidos de nuevo. Sitios como Google Maps usan la localización de tu casa y la actual para proporcionar una información sobre la ruta a seguir y resaltar los negocios locales en los resultados de búsqueda.

Un análisis profundo de los hábitos del usuario se puede usar para anticipar sus intereses y personalizar las respuestas y notificaciones futuras, proporcionando, por ejemplo, una lista de las localizaciones visitadas o populares que querrías buscar en un mapa.

Nota: Vete a Google Maps como usuario anónimo, selecciona el botón Direcciones, e introduce los puntos de partida y destino de un viaje. Ahora inicia sesión en el sistema con tu cuenta de Google, si tienes una (en el panel de abajo aparece información acerca de este proceso donde seleccionas direcciones). El sitio web te permitirá ahora seleccionar las localizaciones de casa y trabajo como puntos de partida y destino (o almacenar estos detalles si no lo has hecho así).

- **Acceso controlado al contenido**

La programación de lado-servidor permite a los sitios restringir el acceso a usuarios autorizados y servir sólo la información que se le permite ver al usuario.

Ejemplos del mundo real incluyen:

- Redes sociales como Facebook permiten a los usuarios controlar totalmente sus propios datos, pero permitiendo sólo a sus amigos ver o comentar sobre ellos. El usuario determina quien puede ver sus datos, y por extensión, los datos de quienes aparecen en sus notificaciones, la autorización es una parte central de la experiencia de usuario.
- La plataforma educativa Moodle controla el acceso al contenido: los artículos son visibles a todos, pero sólo los usuarios que se han identificado pueden editar el contenido. Para comprobar esto, pincha en el botón Edit en la parte superior de esta página — si te has identificado iniciando sesión se te mostrará la vista de edición; si no has iniciado sesión serás enviado a una página de registro.

Nota: Considera otros ejemplos reales donde el acceso al contenido está controlado. Por ejemplo, ¿qué puedes ver si vas al sitio online de tu banco? Inicia sesión con tu cuenta — ¿qué información adicional puedes ver y modificar? ¿Qué información puedes ver y sólo el banco puede cambiar?

- **Almacenar información de sesión/estado**

La programación de lado-servidor permite a los desarrolladores hacer uso de las sesiones — es básicamente un mecanismo que permite al servidor almacenar información sobre el usuario actual del sitio u enviar diferentes respuestas basadas en esa información. Esto permite, por ejemplo, que un sitio sepa que un usuario ha iniciado sesión previamente y presente enlaces a sus correos, o a su historial de órdenes, o quizá guardar el estado de un simple juego de forma que el usuario pueda volver al sitio de nuevo y retomar el juego donde lo dejó.

Nota: Visita el sitio de un periódico que tenga un modelo de suscripción y abre un puñado de pestañas (ej, The Age). Continúa visitando el sitio durante unos pocos días/horas. En algún momento serás finalmente redirigido a las páginas que explican cómo suscribirte y se te impedirá el acceso a los artículos. Esta información es un ejemplo de información de sesión almacenada en cookies.

- **Notificaciones y comunicación**

Los servidores pueden enviar notificaciones de tipo general o específicas de usuario a través del propio sitio web o vía correo electrónico, SMS, mensajería instantánea, conversaciones de video u otros servicios de comunicación.

Unos pocos ejemplos incluyen:

- Facebook y Twitter envían mensajes de correo y SMS para notificarte de nuevas comunicaciones.
- Amazon envía con regularidad emails que sugieren productos similares a aquellos comprados o vistos anteriormente y en los que podrías estar interesado.
- Un servidor web podría enviar mensajes de aviso a los administradores del sistema alertándoles de memoria baja en el servidor o de actividades de usuario sospechosas.

Nota: El tipo de notificación más común es una "confirmación de registro". Elige uno cualquiera de los grandes sitios en que estés interesado (Google, Amazon, Instagram, etc.) y crea una cuenta nueva usando tu dirección de correo. En breve recibirás un email de confirmación de registro, o solicitando un acuse de recibo para activar la cuenta.

- **Análisis de datos**

Un sitio web puede recolectar un montón de datos acerca de los usuarios: qué es lo que buscan, qué compran, qué recomiendan, cuánto tiempo permanecen en cada página. La programación de lado-servidor puede utilizarse para refinar las respuestas basándose en el análisis de estos datos.

Por ejemplo, Amazon y Google anuncian ambos productos basados en búsquedas previas (y adquisiciones).

Aplicaciones Web

Las **aplicaciones web** emplean **páginas web dinámicas** para crear aplicaciones que se **ejecuten en un servidor web** y se muestren en un navegador. Puedes encontrar aplicaciones web para realizar múltiples tareas. Unas de las primeras en aparecer fueron las que viste antes, los clientes de correo, que te permiten consultar los mensajes de correo recibidos y enviar los tuyos propios utilizando un navegador.

Hoy en día existen aplicaciones web para multitud de tareas como procesadores de texto, gestión de tareas, o edición y almacenamiento de imágenes. Estas aplicaciones tienen ciertas ventajas e inconvenientes si las comparas con las aplicaciones tradicionales que se ejecutan sobre el sistema operativo de la propia máquina.

Ventajas de las aplicaciones web:

- **No es necesario instalarlas en aquellos equipos en que se vayan a utilizar.** Se instalan y se ejecutan solamente en un equipo, en el servidor, y esto es suficiente para que se puedan utilizar de forma simultánea desde muchos equipos.
- Como solo se encuentran instaladas en un equipo, **es muy sencillo gestionarlas** (hacer copias de seguridad de sus datos, corregir errores, actualizarlas).
- **Se pueden utilizar en todos aquellos sistemas que dispongan de un navegador web**, independientemente de sus características (no es necesario un equipo potente) o de su sistema operativo.
- **Se pueden utilizar desde cualquier lugar en el que dispongamos de conexión con el servidor.** En muchos casos esto hace posible que se pueda acceder a las aplicaciones desde sistemas no convencionales, como por ejemplo teléfonos móviles.

Inconvenientes de las aplicaciones web:

- **El interface de usuario de las aplicaciones web es la página que se muestra en el navegador.** Esto **restringe** las características del interface a aquellas de una página web.
- **Dependemos de una conexión con el servidor para poder utilizarlas.** Si nos falla la conexión, no podremos acceder a la aplicación web.
- **La información que se muestra en el navegador debe transmitirse desde el servidor.** Esto hace que cierto tipo de aplicaciones no sean adecuadas para su implementación como aplicación web (por ejemplo, las aplicaciones que manejan contenido multimedia, como las de edición de vídeo).

Esta división es así porque el **código que se ejecuta en el cliente web** (en tu navegador) no tiene, o mejor dicho **tradicionalmente no tenía, acceso a los datos** que se almacenan en el **servidor**. Es decir, cuando en tu navegador querías leer un nuevo correo, el código Javascript que se ejecutaba en el mismo no podía obtener de la base de datos el contenido de ese mensaje. La solución era crear una nueva página en el servidor con la información que se pedía y enviarla de nuevo al navegador.

Sin embargo, desde hace unos años existe una **técnica de desarrollo web conocida como AJAX**, que nos posibilita realizar programas en los que el código JavaScript que se ejecuta en el navegador pueda comunicarse con un servidor de Internet para obtener información con la que, por ejemplo, modificar la página web actual.

En nuestro ejemplo, cuando pulsas con el ratón encima de un correo que quieres leer, la página puede contener código Javascript que detecte la acción y, en ese instante, consultar a través de Internet el texto que contiene ese mismo correo y mostrarlo en la misma página, modificando su estructura en caso de que sea necesario. Es decir, sin salir de una página poder modificar su contenido en base a la información que se almacena en un servidor de Internet.

En este módulo vas a aprender a crear aplicaciones web que se ejecuten en el lado del servidor. Otro módulo de este mismo ciclo, Desarrollo Web en Entorno Cliente, enseña las características de la programación de código que se ejecute en el navegador web.

Muchas de las **aplicaciones web actuales utilizan estas dos tecnologías**: la ejecución de **código en el servidor y en el cliente**. Así, el código que se ejecuta en el servidor genera páginas web que ya incluyen código destinado a su ejecución en el navegador. Hacia el final de este módulo verás las técnicas que se usan para programar aplicaciones que incorporen esta funcionalidad.

Programación en entorno servidor.

Cuando programas una **aplicación**, utilizas un **lenguaje de programación**. Por ejemplo, utilizas el lenguaje Java para crear aplicaciones que se ejecuten en distintos sistemas operativos. Al programar cada aplicación utilizas ciertas herramientas como un entorno de desarrollo o librerías

de código. Además, una vez acabado su desarrollo, esa aplicación necesitará ciertos componentes para su ejecución, como por ejemplo una máquina virtual de Java.

Los **componentes principales** con los que debes contar para ejecutar aplicaciones web en un servidor son los siguientes:

- Un **servidor web** para recibir las peticiones de los clientes web (normalmente navegadores) y enviarles la página que solicitan (una vez generada puesto que hablamos de páginas web dinámicas). El servidor web debe conocer el procedimiento a seguir para generar la página web: qué módulo se encargará de la ejecución del código y cómo se debe comunicar con él. Nosotros usaremos el servidor web Apache.
- El **módulo encargado de ejecutar el código** que genera la página web resultante. Este módulo debe integrarse de alguna forma con el servidor web, y dependerá del lenguaje y tecnología que utilicemos para programar la aplicación web, en nuestro caso será el módulo de PHP que debe estar instalado y configurado en el servidor web.
- Una **aplicación de base de datos**, que normalmente también será un servidor, en nuestro caso MySQL. Este módulo, aunque no es estrictamente necesario, si que se utiliza en todas las aplicaciones web para trabajar con datos almacenados en bases de datos.
- El **lenguaje de programación** que utilizarás para desarrollar las aplicaciones, en nuestro caso el lenguaje PHP.

Arquitecturas y plataformas.

La primera elección que harás antes de comenzar a programar una aplicación web es la arquitectura que vas a utilizar. Hoy en día, puedes elegir entre otras:

- Java EE (Enterprise Edition), que antes también se conocía como J2EE. Es una plataforma orientada a la programación de aplicaciones en lenguaje Java. Puede funcionar con distintos gestores de bases de datos, e incluye varias librerías y especificaciones para el desarrollo de aplicaciones de forma modular.

Está apoyada por grandes empresas como Sun y Oracle, que mantienen Java, o IBM. Es una buena solución para el desarrollo de aplicaciones de tamaño mediano o grande. Una de sus principales ventajas es la multitud de librerías existentes en ese lenguaje y la gran base de programadores que lo conocen.

Dentro de esta arquitectura existen distintas tecnologías como las páginas JSP y los servlets, ambos orientados a la generación dinámica de páginas web, o los EJB, componentes que normalmente aportan la lógica de la aplicación web.

- AMP. Son las siglas de Apache, MySQL y PHP/Perl/Python. Las dos primeras siglas hacen referencia al servidor web (Apache) y al servidor de base de datos (MySQL). La última se corresponde con el lenguaje de programación utilizado, que puede ser PHP, Perl o Python, siendo PHP el más empleado de los tres.

Dependiendo del sistema operativo que se utilice para el servidor, se utilizan las siglas LAMP (para Linux), WAMP (para Windows) o MAMP (para Mac). También es posible usar otros componentes, como el gestor de bases de datos PostgreSQL en lugar de MySQL.

Todos los componentes de esta arquitectura son de código libre (open source). Es una plataforma de programación que permite desarrollar aplicaciones de tamaño pequeño o mediano con un aprendizaje sencillo. Su gran ventaja es la gran comunidad que la soporta y la multitud de aplicaciones de código libre disponibles.

- CGI/Perl. Es la combinación de dos componentes: Perl, un potente lenguaje de código libre creado originalmente para la administración de servidores, y CGI, un estándar para permitir al servidor web ejecutar programas genéricos, escritos en cualquier lenguaje (también se pueden utilizar por ejemplo C o Python), que devuelven páginas web (HTML) como resultado de su ejecución. Es la más primitiva de las arquitecturas que comparamos aquí.

El principal inconveniente de esta combinación es que CGI es lento, aunque existen métodos para acelerarlo. Por otra parte, Perl es un lenguaje muy potente con una amplia comunidad de usuarios y mucho código libre disponible.

- ASP.Net es la arquitectura comercial propuesta por Microsoft para el desarrollo de aplicaciones. Es la parte de la plataforma .Net destinada a la generación de páginas web dinámicas. Proviene de la evolución de la anterior tecnología de Microsoft, ASP.

El lenguaje de programación puede ser Visual Basic.Net o C#. La arquitectura utiliza el servidor web de Microsoft, IIS, y puede obtener información de varios gestores de bases de datos entre los que se incluye, como no, Microsoft SQL Server.

Una de las mayores ventajas de la arquitectura .Net es que incluye todo lo necesario para el desarrollo y el despliegue de aplicaciones. Por ejemplo, tiene su propio entorno de desarrollo, Visual Studio, aunque hay otras opciones disponibles. La mayor desventaja es que se trata de una plataforma comercial de código propietario.

Código embebido en lenguajes de marca

Cuando la web comenzó a evolucionar desde las páginas web estáticas a las dinámicas, una de las primeras tecnologías que se utilizaron fue la ejecución de código utilizando CGI. Los guiones CGI son programas estándar, que se ejecutan por el sistema operativo, pero que generan como salida el código HTML de una página web. Por tanto, los guiones CGI deben contener, mezcladas dentro de su código, sentencias encargadas de generar la página web.

Esta es una de las principales formas de realizar páginas web dinámicas: **integrar las etiquetas HTML en el código de los programas**. Es decir, los programas como el guión anterior incluyen dentro de su código sentencias de salida, que son las que forman el código HTML de la página web que se obtendrá cuando se ejecuten. De esta forma se programan, por ejemplo, los guiones CGI y los servlets.

Un enfoque distinto consiste en **integrar el código del programa en medio de las etiquetas HTML de la página web**. De esta forma, el contenido que no varía de la página se puede introducir directamente en HTML, y el lenguaje de programación se utilizará para todo aquello que pueda variar de forma dinámica.

Por ejemplo, puedes incluir dentro de una página HTML un pequeño código en lenguaje PHP que muestre el nombre del servidor o alguna fecha calculada.

Esta metodología de programación es la que se emplea en los lenguajes ASP, **PHP** y con las páginas JSP de Java EE.

2. Introducción PHP

PHP (Personal Home Page) es, según el índice PYPL (PopularitY of Programming Language index), el segundo lenguaje de programación más utilizado en el mundo, únicamente superado por Java¹.

PHP es un lenguaje de programación estructurado y, como tal, hace uso de variables, sentencias condicionales, bucles, funciones... PHP es también un lenguaje de programación orientado a objetos y, por consiguiente, permite definir clases con sus métodos correspondientes y crear instancias de esas clases.

Es muy frecuente combinar PHP con HTML y Javascript por lo que, para sacar provecho del libro, es recomendable tener unos conocimientos básicos sobre estas materias. Una web excelente para aprender HTML y Javascript es W3Schools.com. La documentación de php la podéis consultar en php.net, una web con todas las funciones disponibles en el lenguaje, muy bien explicadas y con ejemplos.

A diferencia de JavaScript o HTML que se ejecutan en el navegador, PHP se ejecuta en el servidor, por eso nos permite acceder a los recursos que tiene el servidor, por ejemplo a una base de datos. Los programas escritos en PHP se ejecutan en el servidor y el resultado se envía al navegador. El resultado es normalmente una página HTML.

Al ser PHP un lenguaje que se ejecuta en el servidor no es necesario que el navegador lo soporte, es independiente del navegador, pero sin embargo para que las páginas PHP funcionen, el servidor donde están alojadas debe soportar PHP. Hoy en día la práctica totalidad de servidores que ofrecen servicios de hosting soportan PHP por defecto.

Los programas necesarios para probar los ejemplos de este libro y para realizar los ejercicios son los siguientes:

1. Un navegador web
2. Un editor de texto
3. El lenguaje PHP
4. Un servidor web (p. ej. Apache)
5. El módulo de PHP para el servidor
6. Un gestor de bases de datos como por ejemplo MySQL.

Afortunadamente hay paquetes que ya incluyen un entorno con todos los componentes necesarios para funcionar tras su instalación; todo ello convenientemente configurado y preparado para empezar a programar sin tener que preocuparnos de nada que no sea nuestro programa PHP. Nosotros usaremos XAMPP, tan solo tenéis que descargarlo de su página oficial e instalar con las opciones por defecto.

¹

Visual Studio Code

Como editor de código vamos a usar 'Visual Studio Code', aunque si alguien tiene preferencia por otro editor como 'Sublime text' puede usarlo perfectamente. Los plugins que deberías instalar para trabajar con mayor comodidad son los siguientes:

- Spanish Language Pack for Visual Studio Code
- HTML CSS Support
- PHP Getters & Setters
- PHP Intelephense
- Prettier – Code formatter
- Vscod-icons
- Laravel Blade Snippets
- Laravel Snippets
- Open PHP/HTML/JS In Browser [*1](#)
- MySQL de cweijan (delfín con fondo azul) [*2](#)
- PHP Awesome Snippets [*3](#)
- PHP Debug [*4](#)

[*1](#). Este plugin permite abrir el archivo php directamente en el navegador desde visual studio code, si no funciona correctamente configurar el parámetro documentRootFolder con la ubicación de la carpeta htdocs y el parámetro urlToOpen con "http://localhost/", aunque con el navegador Edge no va bien.

[*2](#). Este plugin permite modificar los datos y probar sentencias SQL en VsCode sin abrir PhpMyadmin, aunque para la creación de las bases de datos y sus tablas es mejor hacerlo en PhpMyadmin.

[*3](#). Este plugin tiene predefinido multitud de snippets que nos permiten autocompletar código mientras estamos editando, ahorrándonos tiempo de edición. Aunque podemos definir nuestros propios snippets. Por ejemplo, para autocompletar la etiqueta php cuando estamos en el lenguaje html, seguir los siguientes pasos:

1. Configuración -> Paleta de comandos...

2. Escribir snippet y pulsar enter

3. Escribir html y pulsar enter

4. Añadir en el archivo 'html.json' que se ha abierto, antes de cerrar la última llave '}' el código:

```
"php": {  
  "prefix": "php",  
  "body": [  
    "<?php $1 ?>"  
  ],  
  "description": "php tag"  
}
```

Para crear otros snippets dentro del lenguaje php, que permitan autocompletar un texto definido por nosotros hay que configurarlos en el archivo 'php.json' de forma similar a la anterior.

NOTA: A veces cuando estamos editando no nos aparece el desplegable con las opciones de autocompletado, para solucionarlo debemos pulsar control+espacio y nos saldrá la lista de autocompletado, lo que nos permitirá codificar de manera más eficiente.

*4. Este plugin permite depurar la ejecución del código php, pero para que sea posible hay que realizar una configuración consistente en:

1. Después de instalar el plugin, pulsar en el enlace “[Xdebug installation wizard](#)” de la descripción del plugin y en la ventana que aparece en la página pegar todo el texto que muestra la función phpinfo (ejecutar en una página ‘echo phpinfo();’) y pulsar el botón que hay debajo de la ventana.

Nota: tened cuidado de que el navegador no traduzca automáticamente la salida de phpinfo()

2. Descargar el archivo dll que ha generado (si indicara que la versión de php no es compatible, instalar de nuevo xampp con la última versión de php y repetir el proceso), renombrarlo con el nombre ‘php_xdebug.dll’ y ubicarlo en la carpeta ‘C:\xampp\php\ext’

3. Añadir al final del archivo php.ini (podemos abrirlo desde el botón config de Apache del panel Xampp) las siguientes líneas y después reiniciar Apache:

```
zend_extension = xdebug
xdebug.mode = debug
xdebug.start_with_request = yes
```

4. Para depurar solo tenemos que pulsar el icono de depuración del lateral izquierdo de VsCode, crear automáticamente el archivo json sugerido, cerrarlo y ya podemos iniciar la depuración pulsando F5 o con el triangulito verde (añadir previamente los puntos de ruptura deseados).

Puede que tengáis que añadir en el archivo settings.json de VsCode el parámetro:

```
"php.debug.executablePath": "C:/xampp/php/php.exe"
```

Si queréis aprender más sobre este editor, tal como atajos de teclado, configuración, etc, podéis visitar el tutorial <https://www.tutorialesprogramacionya.com/herramientas/vscodeya/> o cualquiera de los tutoriales que hay por internet.

3. Conceptos básicos: Variables, Operadores y formularios

Integración de PHP en HTML ¡Hola mundo!

Para probar la ejecución de nuestro primer programa, tendremos que crear un archivo web, por ejemplo index.html dentro de la carpeta pública del servidor web, normalmente c:/xampp/htdocs/. Para probar su ejecución debes iniciar el servidor web y realizar una petición como cliente a dicho servidor, por lo que deberás cargar la ruta http://localhost en la barra de direcciones, si en htdocs existe un archivo index.html o index.php se mostrará en el navegador, si no, se mostrará un listado con los archivos de la carpeta htdocs, para cargarlos en el navegador, hacer click sobre ellos. Ejemplo nuestro primer programa en php:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      echo "¡Hola mundo!";
    ?>
  </body>
</html>
```

Como habrás observado, el programa contiene código en HTML mezclado con una sentencia en PHP. Cada vez que quieras insertar código en PHP, deberás encerrarlo entre las etiquetas <?php y ?>. En caso de que todo el código del fichero sea PHP y no haya nada de HTML, se indica únicamente la etiqueta de inicio <?php.

La instrucción `echo` sirve para volcar texto en la página HTML. No es necesariamente el texto que se quiere mostrar, sino el que queremos que reciba el navegador para interpretar la página html. Una prueba interesante en estos primeros pasos consiste en ver el código fuente que se ha generado desde el navegador, donde comprobaremos el código html resultante que ha recibido el navegador, sin una sola sentencia php. Veamos otro ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Utilizo PHP para poner en negrita una palabra -->
```

```
Hola <?php echo "<b><u>"; ?> mundo <?php echo "</u></b>"; ?>
</body>
</html>
```

Observa que esta vez, `echo` ha servido para volcar en HTML las etiquetas `` y `<u>` que hacen que una cadena de caracteres se muestre en negrita y en cursiva respectivamente. Fíjate que después de una sentencia en PHP se escribe un punto y coma.

Aquí tienes otro ejemplo, en este caso mostramos una línea utilizando HTML y otra utilizando PHP:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <!-- Muestra una frase con HTML -->
    Hola mundo<br>
    <!-- Muestra una frase con PHP -->
    <?php
      echo "Es muy fácil programar en PHP.";
    ?>
  </body>
</html>
```

3.1 Variables

Definición de variables

Una variable es un contenedor de información, es algo así como una cajita que tiene un nombre y en la que podemos meter un valor. Las variables pueden almacenar números enteros, números decimales, caracteres, cadenas de caracteres (palabras o frases), etc. El contenido de las variables se puede mostrar y se puede cambiar durante la ejecución de una página PHP (por eso se llaman variables).

Los nombres de las variables en php comienzan con el símbolo del dólar (\$) y no es necesario definirlos como se hace en otros lenguajes de programación como C, Java, Pascal, etc. La misma variable puede contener un número y luego el nombre de una ciudad, no existe restricción en cuanto al tipo como en la mayoría de los lenguajes.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
```



```

<body>
  <?php
    $x = 24;
    $pi = 3.1416;
    $animal = "conejo";
    $saludo = "hola caracola";
    echo $x, "<br>", $pi, "<br>", $animal, "<br>", $saludo;
  ?>
</body>
</html>

```

En este ejemplo se han definido las variables `$x`, `$pi`, `$animal` y `$saludo`. Con la instrucción `echo` se ha mostrado el valor que contienen, insertando un salto de línea entre ellas. Fíjate que la coma sirve para unir trozos de una cadena de caracteres.

El tipo de la variable se determina por el contexto en el cual se emplea la variable. Es decir, si se asigna un valor string a una variable `$var`, entonces `$var` se convierte en un string. Si un valor integer es entonces asignado a la misma variable `$var`, ésta se convierte en integer.

Un ejemplo de la conversión de tipos automática de PHP es el operador suma '+'. Si al menos uno de los operandos es float, entonces ambos operandos son evaluados como floats y el resultado de la expresión será un float. De otra manera, los operandos serán interpretados como integers, y el resultado será entonces integer. Tenga en cuenta que esto no implica que se cambien los tipos de los propios operandos, sino solo del resultado de la expresión. Si alguna conversión automática no es posible provocará un error (por ejemplo `3 + "hola"`), y en otras ocasiones puede provocar un warning, aunque la conversión sea posible (por ejemplo `3 + "2hola"` resulta 5, ignorando las letras del segundo operando).

```

<?php
$foo = "0"; // $foo es string (ASCII 48)
$foo = 2; // $foo es ahora un integer (2)
$foo = $foo + 1.3; // $foo es ahora un float (3.3)
$foo = 5 + "10 caracolas"; // $foo es ahora un integer (15)
?>

```

Forzado de tipos: El forzado de tipos en PHP funciona de la misma manera que en C, donde el nombre del tipo deseado se escribe entre paréntesis antes de la variable que se quiera forzar. Los siguientes forzados de tipos primitivos están permitidos:

- (int), (integer) - forzado a integer
- (bool), (boolean) - forzado a boolean, 0 o null -> false, y resto de valores -> true
- (float), (double), (real) - forzado a float
- (string) - forzado a string

```

<?php
$foo = 10;    // $foo es un integer
$bar = (boolean) $foo;    // $bar es un boolean con valor true
?>

```

3.2 Operadores aritméticos

Los operadores de PHP son similares a los de cualquier otro lenguaje de programación. Estos son los operadores que se pueden aplicar tanto a las variables como a las constantes numéricas:

Operador	Nombre	Ejemplo	Descripción
+	suma	20 + \$x	suma dos números
-	resta	\$a - \$b	resta dos números
*	multiplicación	10 * 7	multiplica dos números
/	división	\$altura / 2	divide dos números
%	módulo	5 % 2	devuelve el resto de la división entera
++	incremento	\$a++	incrementa en 1 el valor de la variable
--	decremento	\$a--	decrementa en 1 el valor de la variable

A continuación, se muestra un programa que ilustra el uso de estos operadores:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = 8; $b = 3;
      echo $a + $b, "<br>";
      echo $a - $b, "<br>";
      echo $a * $b, "<br>";
      echo $a / $b, "<br>";
      $a++;
      echo $a, "<br>";
      $b--;
      echo $b, "<br>";
    ?>
  </body>
</html>
```

Mientras depuramos un programa, con frecuencia necesitamos ver el valor de las variables. Puedes hacer `echo` sobre cada una de ellas como hemos visto en los ejemplos anteriores pero es muy cómodo usar `print_r(get_defined_vars());` que muestra el valor de todas y cada una de las variables que se han definido.

```
<!DOCTYPE html>
<html>
  <head>
```

```

<meta charset="UTF-8">
</head>
<body>
    <?php
        $numero = 20;
        $palabra = "hola";
        print_r(get_defined_vars());
    ?>
</body>
</html>

```

3.3 Ejercicios para practicar



Ejercicio 1

Escribe un programa que muestre tu nombre por pantalla. Utiliza código PHP.



Ejercicio 2

Modifica el programa anterior para que muestre tu dirección y tu número de teléfono. Cada dato se debe mostrar en una línea diferente. Mezcla de alguna forma las salidas por pantalla, utilizando tanto HTML como PHP.



Ejercicio 3

Escribe un programa que muestre por pantalla 10 palabras en inglés junto a su correspondiente traducción al castellano. Las palabras deben estar distribuidas en dos columnas. Utiliza la etiqueta `<table>` de HTML.



Ejercicio 4

Escribe un programa que muestre tu horario de clase mediante una tabla. Aunque se puede hacer íntegramente en HTML (igual que los ejercicios anteriores), ve intercalando código HTML y PHP para familiarizarte con éste último.



Ejercicio 5

Escribe un programa que utilice las variables `$x` y `$y`. Asigne los valores 144 y 999 respectivamente. A continuación, muestra por pantalla el valor de cada variable, la suma, la resta, la división y la multiplicación.



Ejercicio 6

Crea la variable `$nombre` y asígnale tu nombre completo. Muestra su valor por pantalla de tal forma que el resultado sea el mismo que el del ejercicio 1.



Ejercicio 7

Crea las variables `$nombre`, `$direccion` y `$telefono` y asígneles los valores adecuados. Muestra los valores por pantalla de tal forma que el resultado sea el mismo que el del ejercicio 2.



Ejercicio 8

Realiza un conversor de euros a pesetas. La cantidad en euros que se quiere convertir deberá estar almacenada en una variable.



Ejercicio 9

Realiza un conversor de pesetas a euros. La cantidad en pesetas que se quiere convertir deberá estar almacenada en una variable.



Ejercicio 10

Escribe un programa que pinte por pantalla una pirámide rellena a base de asteriscos. La base de la pirámide debe estar formada por 9 asteriscos.



Ejercicio 11

Igual que el programa anterior, pero esta vez la pirámide estará hueca (se debe ver únicamente el contorno hecho con asteriscos).



Ejercicio 12

Igual que el programa anterior, pero esta vez la pirámide debe aparecer invertida, con el vértice hacia abajo.

3.4 Recogida de datos por teclado mediante formularios

A. Recogida de datos en dos pasos

En una página web, los datos se introducen mediante formularios. En la mayoría de las ocasiones tendremos dos páginas: una página con un formulario que recoge los datos y otra página que los procesa.

A continuación, tenemos una página con nombre `index.php` (se podría llamar también `index.html` ya que no contiene código PHP):

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Introduce tu nombre:
    <form action="saluda.php" method="get">
      <input type="text" name="nombre"><br>
      <input type="submit" value="Enviar">
    </form>
  </body>
</html>
```

Al pulsar el botón Enviar, el contenido del formulario se envía a la página que indicamos en el atributo `action`. La página `saluda.php` tendría el siguiente código:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Hola <?php echo $_GET['nombre'] ?>, que tengas un buen día.
  </body>
</html>
```

Observa que la información que se quiere enviar se introduce en un cuadro de texto dentro del formulario (página `index.php`). A esa información la llamamos en este caso `nombre`. Para recoger esa información se utiliza `$_GET['nombre']`.

Cuando los datos que se recogen y se manipulan son números en lugar de cadenas de caracteres, el procedimiento es el mismo.

En el siguiente ejemplo tenemos una aplicación que suma dos números. El programa `index.php` recoge los datos y el programa `suma.php` suma los dos números que recibe y muestra el resultado.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <h3>Calcula la suma de dos números</h3>
    <form action="suma.php" method="get">
      Primer número: <input type="number" name="a"><br>
      Segundo número: <input type="number" name="b"><br>
      <input type="submit" value="Sumar">
    </form>
  </body>
</html>
```

Fíjate que los campos `input` del formulario son de tipo `number`. Se podrían declarar de tipo `text` y el programa funcionaría sin problemas pero se declaran `number` para que el formulario no deje pasar los datos si se introduce algo que no sea un número. Imagínate que el usuario introduce la palabra `hola` en un campo en lugar de un número. Si el `input` es de tipo `text`, la palabra `hola` llegaría a `suma.php` y se intentaría sumar, lo que provocaría un error. Sin embargo, si el `input` es de tipo `number`, el formulario no envía `hola` sino que muestra un mensaje diciéndole al usuario que introduzca un número.

En la medida de lo posible, intenta utilizar siempre el tipo adecuado en los campos `input`; por ejemplo `number`, `color`, `date`, `email`, `url`, etc. de tal forma que los datos que envía el formulario tengan el formato correcto.

A continuación, se muestra el fichero `suma.php`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = $_GET['a'];
      $b = $_GET['b'];
      echo "La suma de $a mas $b es ", $a + $b;
    ?>
  </body>
</html>
```

Como hemos visto anteriormente, en PHP no hace falta declarar las variables especificando el tipo. Una misma variable puede contener incluso valores de distintos tipos durante su vida útil. Esto hace que no necesitemos ninguna conversión como habría que hacer utilizando otros lenguajes.

B. Métodos `get` y `post`

Mediante el atributo `method` de la etiqueta `form` se debe especificar un método de envío; los dos métodos posibles son `get` y `post`. El resultado final es el mismo, la única diferencia radica en que con el método `get` se pueden ver los parámetros que envía el formulario en la barra de direcciones del navegador.

3.5 Ejercicios para practicar

Ejercicio 1

Realiza un programa que pida dos números y luego muestre el resultado de su multiplicación.

Ejercicio 2

Realiza un conversor de euros a pesetas. Ahora la cantidad en euros que se quiere convertir se deberá introducir por teclado.

Ejercicio 3

Realiza un conversor de pesetas a euros. La cantidad en pesetas que se quiere convertir se deberá introducir por teclado.

Ejercicio 4

Escribe un programa que sume, reste, multiplique y divida dos números introducidos por teclado.

Ejercicio 5

Escribe un programa que calcule el área de un rectángulo.

Ejercicio 6

Escribe un programa que calcule el área de un triángulo.

Ejercicio 7

Escribe un programa que calcule el total de una factura a partir de la base imponible.

Ejercicio 8

Escribe un programa que calcule el salario semanal de un trabajador en base a las horas trabajadas. Se pagarán 12 euros por hora.

Ejercicio 9

Escribe un programa que calcule el volumen de un cono mediante la fórmula $V = \frac{1}{3}\pi r^2 h$.

Ejercicio 10

Realiza un conversor de Mb a Kb.

Ejercicio 11

Realiza un conversor de Kb a Mb.

3.6 Conceptos de la Unidad

- **Variables:** Los nombres de las variables comienzan con el símbolo dólar (\$) y no es necesario definirlos. La misma variable puede contener distintos tipos de dato.
La función `var_dump(vble1, vble2, ...)` imprime el tipo y valor de cada variable o array pasado por parámetro.
La función `print_r(vble)` imprime sólo el valor de una variable o array pasado por parámetro.
La función `isset(vble)` devuelve verdadero si la vble tiene definido algún valor.
- **Constantes:** Definir las constantes en mayúsculas y sin el símbolo '\$':

```
define("PI", 3.14); -> echo "<p>El valor de pi es PI</p>\n";
```

Algunas constantes predefinidas:

`M_PI`: valor de PI

`PHP_INT_MAX`: valor máximo de un entero

`PHP_INT_MIN`: valor mínimo de un entero

- **Operadores aritméticos**

Operador	Nombre	Ejemplo	Descripción
+	suma	20 + \$x	suma dos números
-	resta	\$a - \$b	resta dos números
*	multiplicación	10 * 7	multiplica dos números
/	división	\$altura / 2	divide dos números
%	módulo	5 % 2	devuelve el resto de la división entera
++	incremento	\$a++	incrementa en 1 el valor de \$a
--	decremento	\$a--	decrementa en 1 el valor de \$a

Si ++ o -- se pone delante de la variable en una expresión se ejecuta antes de resolverla.

`rand(mínimo, máximo)` genera un aleatorio entero entre dos valores.

`round(valor, nºdecimales)` redondea el valor con el número de decimales indicado

`floor(numero con decimales)` devuelve la parte entera sin decimales redondeando

`ceil(numero con decimales)` devuelve la parte entera redondeando hacia arriba siempre

`intval(numero con decimales)` devuelve la parte entera ignorando los decimales (truncar)

- **Operadores para String**

`.` concatena dos Strings

`.=` añade al final del String de la izquierda, el String de la derecha

`""` delimitan un String y amplían el valor de variables primitivas en su interior

" delimitan un String y NO amplían el valor de variables primitivas en su interior
\
hacen escapar el comportamiento de caracteres especiales dentro de un string
Se pueden combinar comillas dobles dentro de simples y viceversa.

- Recogida de datos de formularios (si no se define method, por defecto se usa GET):
\$_GET['nombre'] : (array asociativo para método get y parámetros por url ->
Mi enlace
\$_POST['nombre'] : (array asociativo para método post. OBLIGATORIO con sesiones)
\$_REQUEST['nombre'] : (array asociativo para métodos get, post y cookies)
enctype = "multipart/form-data" : en un formulario indica que se envía texto y ficheros (es obligatorio usar método post)
<input type="": usar los tipos adecuados (number, color, date, email, url ...), para restringir los tipos de datos enviados a la página destino.
- Envío de datos por URL, en vez de usar formularios
Mi enlace
- Redireccionamiento automático a otra página:
header('location:paginadestino.php');
- Redireccionamiento a la última página visitada antes de la actual
header('location:'.getenv('HTTP_REFERER'));

Si al volver atrás con el navegador a una página con formularios nos salta error de que la página ha caducado podemos solucionar el error evitando que el navegador almacene en caché la respuesta, añadiendo estas líneas al principio:

```
header("Cache-Control: no-store, no-cache, must-revalidate, max-age=0");  
header("Cache-Control: post-check=0, pre-check=0", false);  
header("Pragma: no-cache");  
header("Expires: Sat, 01 Jan 2000 00:00:00 GMT");
```

3.7 Ejercicios para resolver

EJERCICIO Nº 1

Diseñar un formulario web que pida la altura y el diámetro de un cilindro. Una vez el usuario introduzca los datos y pulse el botón calcular, deberá calcularse el volumen del cilindro y mostrarse el resultado en el navegador. Mostrar la imagen de un cilindro junto al resultado y un título "Calculo del volumen de un cilindro", intenta dar un aspecto agradable a la página de resultado.

EJERCICIO Nº 2

Diseñar una web para jugar a la lotería primitiva. En un formulario se pedirá introducir la combinación de 6 números entre 1 y 49 y el número de serie entre 1 y 999. Mostrar la combinación generada y la introducida por el usuario en dos filas de una tabla.

EJERCICIO Nº 3

Mostrar en una página varios parámetros para configurar el aspecto de otra página: color de fondo, tipo de letra, alineación del texto, imagen del banner (entre 3 posibles), y demás reglas de estilo que se te ocurran. Una vez enviada la información se mostrará una página con el contenido que desees acorde a los estilos elegidos.

EJERCICIO Nº 4

Diseñar un desarrollo web simple con php que pida al usuario el precio de un producto en tres establecimientos distintos denominados "Tienda 1", "Tienda 2" y "Tienda 3". Una vez se introduzca esta información se debe calcular y mostrar el precio medio del producto en las tres tiendas. Mostrar en la página resultado, una tabla con un título, el precio en cada una de las tiendas, la media de los tres precios y la diferencia del precio de cada tienda con la media. Combina celdas para que quede visualmente agradable.

EJERCICIO Nº 5

Diseñar un desarrollo web simple con PHP que dé respuesta a la necesidad que se plantea a continuación:

Un operario de una fábrica recibe cada cierto tiempo un depósito cilíndrico de dimensiones variables, que debe llenar de aceite a través de una toma con cierto caudal disponible. Se desea crear una aplicación web que le indique cuánto tiempo transcurrirá hasta el llenado del depósito. Para calcular dicho tiempo el usuario introducirá los siguientes datos: diámetro y altura del cilindro y caudal de aceite (litros por minuto). Una vez introducidos se mostrará el tiempo total en horas y minutos que se tardará en llenar el cilindro.

4. Estructuras de control

4.1 Condicionales

A. Sentencia `if`

La sentencia `if` permite la ejecución de una serie de instrucciones dependiendo del resultado de evaluar una expresión lógica¹. Se podría traducir al español como “si se cumple esta condición haz esto y si no, haz esto otro”.

El formato del `if` es el siguiente:

```
if (condición) {  
    Sentencias a ejecutar cuando la condición es cierta  
} else {  
    Sentencias a ejecutar cuando la condición es falsa.  
}
```

A continuación, se muestra un ejemplo completo en PHP:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <?php  
      $a = 8;  
      $b = 3;  
      if ($a < $b) {  
        echo "a es menor que b";  
      }else{  
        echo "a no es menor que b";  
      }  
    ?>  
  </body>  
</html>
```

En este ejemplo, la condición no es verdadera por lo que se ejecuta la parte de código correspondiente al `else`.

En PHP también existe la posibilidad de utilizar una condición con la pareja de símbolos `?` y `:` al estilo del lenguaje C, el formato es el siguiente:

¹ El resultado de evaluar una expresión lógica siempre es verdadero o falso.

(condición) ? expresión1 : expresión2

El funcionamiento es el siguiente: se evalúa la condición; si la condición es verdadera, el resultado que se devuelve es `expresión1` y si la condición es falsa, se devuelve `expresión2`.

A continuación, se muestra un programa que ilustra este tipo de sentencia.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      $x = (20 > 6) ? 11 : 22;
      echo $x;
    ?>
  </body>
</html>
```

La expresión `(20 > 6)` es verdadera, por tanto, en la variable `$x` se almacena el número `11` que es lo que se muestra por pantalla. Prueba a cambiar la condición para que sea falsa para comprobar que, esta vez, se muestra el `22`.

B. Operadores de comparación

En el ejemplo de la sección anterior se utiliza el operador `<` en la comparación `if ($a < $b)` para saber si el valor de la variable `$a` es menor que el de la variable `$b`. Existen más operadores de comparación, a continuación, se muestran todos ellos en una tabla:

Operador	Nombre	Ejemplo	Devuelve verdadero cuando...
<code>==</code>	Igual	<code>\$a == \$b</code>	\$a es igual \$b (aunque sean de diferente tipo)
<code>===</code>	Igual	<code>\$a === \$b</code>	\$a es igual \$b (y además son del mismo tipo)
<code>!=</code>	Distinto	<code>\$a != \$b</code>	\$a es distinto \$b
<code><</code>	Menor que	<code>\$a < \$b</code>	\$a es menor que \$b
<code>></code>	Mayor que	<code>\$a > \$b</code>	\$a es mayor que \$b
<code><=</code>	Menor o igual	<code>\$a <= \$b</code>	\$a es menor o igual que \$b
<code>>=</code>	Mayor o igual	<code>\$a >= \$b</code>	\$a es mayor o igual que \$b
<code><=></code>	Nave espacial	<code>\$a <=> \$b</code>	-1 si \$a es menor, 0 si son iguales y 1 si \$b es menor

En el siguiente programa se muestra claramente la diferencia entre `==` y `===`.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
```

```

<body>
  <?php
    $a = 5;
    $b = "5";
    echo ($a == 5)?"verdadero":"falso", "<br>";
    echo ($a === 5)?"verdadero":"falso", "<br>";
    echo ($b == 5)?"verdadero":"falso", "<br>";
    echo ($b === 5)?"verdadero":"falso", "<br>";
  ?>
</body>
</html>

```

El resultado del programa sería el siguiente:

```
verdadero verdadero verdadero falso
```

C. Operador null coalescente (??)

Desde PHP 7.0, el operador ?? es una forma corta de comprobar si un valor existe y no es null para tomarlo en una expresión o en caso contrario tomar un valor por defecto.

```

$variable = $valor1 ?? $valor2;
// Si $valor1 está definido y no es null, se asigna a $variable.
// Si $valor1 no está definido o es null, se asigna $valor2.

```

Ejemplo:

```

$nombre = $_GET['nombre'] ?? 'Anónimo';
echo $nombre;
// Si 'nombre' no está en la solicitud GET o es null, se mostrará 'Anónimo'.

```

D. Operadores lógicos

Las comparaciones se pueden combinar mediante los operadores lógicos. Por ejemplo, si queremos saber si la variable \$a es mayor que \$b y además menor que \$c, escribiríamos `if (($a > $b) && ($a < $c))`. La siguiente tabla muestra los operadores lógicos de PHP:

Operador	Nombre	Ejemplo	Devuelve verdadero cuando...
&&	Y	(7>2) && (2<4)	Devuelve verdadero cuando ambas condiciones son verdaderas.
and	Y	(7>2) and (2<4)	Devuelve verdadero cuando ambas condiciones son verdaderas.
	O	(7>2) (2<4)	Devuelve verdadero cuando al menos una de las dos es verdadera.
or	O	(7>2) or (2<4)	Devuelve verdadero cuando al menos una de las dos es verdadera.
!	No	! (7>2)	Niega el valor de la expresión.

A continuación, se muestra el uso de estos operadores lógicos en un programa PHP:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $a = 8;
      $b = 3;
      $c = 3;

      echo ($a == $b) && ($c > $b), "<br>";
      echo ($a == $b) || ($b == $c), "<br>";
      echo !($b <= $c), "<br>";
    ?>
  </body>
</html>
```

Observa que cuando el resultado de evaluar la expresión lógica es verdadero, se muestra un 1 por pantalla.

E. Sentencia switch

En ocasiones es necesario comparar el valor de una variable con una serie de valores concretos. Es algo parecido (aunque no exactamente igual) a utilizar varios `if` consecutivos.

El formato del `switch` es el siguiente:

```
switch(variable) {
  case valor1: sentencias break;
  case valor2: sentencias break;
  ...
  default:
    sentencias
}
```

A continuación, se muestra un ejemplo completo en PHP:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php $posicion = "arriba";
    switch($posicion) {
      case "arriba": // Bloque 1
        echo "La variable contiene";
        echo " el valor arriba";
        break;
    }
  </body>
</html>
```

```
    case "abajo": // Bloque 2
        echo "La variable contiene";
        echo " el valor abajo";
        break;
    default: // Bloque 3
        echo "La variable contiene otro valor";
        echo " distinto de arriba y abajo";
    }
?>
</body>
</html>
```

Prueba a cambiar el valor de la variable `$posicion` del ejemplo anterior y observa cómo se ejecuta el bloque de sentencias correspondiente.

4.2 Ejercicios para practicar

Ejercicio 1

Escribe un programa que pida por teclado un día de la semana y que diga qué asignatura toca a primera hora ese día.

Ejercicio 2

Realiza un programa que pida una hora por teclado y que muestre luego buenos días, buenas tardes o buenas noches según la hora. Se utilizarán los tramos de 6 a 12, de 13 a 20 y de 21 a 5, respectivamente. Sólo se tienen en cuenta las horas, los minutos no se deben introducir por teclado.

Ejercicio 3

Escribe un programa en que dado un número del 1 a 7 escriba el correspondiente nombre del día de la semana.

Ejercicio 4

Vamos a ampliar uno de los ejercicios de la relación anterior para considerar las horas extras. Escribe un programa que calcule el salario semanal de un trabajador teniendo en cuenta que las horas ordinarias (40 primeras horas de trabajo) se pagan a 12 euros la hora. A partir de la hora 41, se pagan a 16 euros la hora.

Ejercicio 5

Realiza un programa que resuelva una ecuación de primer grado (del tipo $ax + b = 0$).

Ejercicio 6

Realiza un programa que calcule el tiempo que tardará en caer un objeto desde una altura h . Aplica

la fórmula $t = \sqrt{\frac{2h}{g}}$ siendo $g = 9.81 m/s^2$

Ejercicio 7

Realiza un programa que calcule la media de tres notas.

Ejercicio 8

Amplía el programa anterior para que diga la nota del boletín (insuficiente, suficiente, bien, notable o sobresaliente).

Ejercicio 9

Realiza un programa que resuelva una ecuación de segundo grado (del tipo $ax^2 + bx + c = 0$).

Ejercicio 10

Escribe un programa que nos diga el horóscopo a partir del día y el mes de nacimiento.

Ejercicio 11

Escribe un programa que dada una hora determinada (horas y minutos), calcule los segundos que faltan para llegar a la medianoche.

Ejercicio 12

Realiza un minicuestionario con 10 preguntas tipo test sobre las asignaturas que se imparten en el curso. Cada pregunta acertada sumará un punto. El programa mostrará al final la calificación obtenida. Pásale el minicuestionario a tus compañeros y pídeles que lo hagan para ver qué tal andan de conocimientos en las diferentes asignaturas del curso.

Ejercicio 13

Escribe un programa que ordene tres números enteros introducidos por teclado.

Ejercicio 14

Realiza un programa que diga si un número introducido por teclado es par y/o divisible entre 5.

Ejercicio 15

Realiza un programa que nos diga si hay probabilidad de que nuestra pareja nos está siendo infiel. El programa irá haciendo preguntas que el usuario contestará con verdadero o falso. Puedes utilizar radio buttons. Cada pregunta contestada como verdadero sumará 3 puntos. Las preguntas contestadas con falso no suman puntos. Utiliza el fichero test_infidelidad.txt para obtener las preguntas y las conclusiones del programa.

Ejercicio 16

Escribe un programa que diga cuál es la última cifra de un número entero introducido por teclado.

Ejercicio 17

Escribe un programa que diga cuál es la primera cifra de un número entero introducido por teclado. Se permiten números de hasta 5 cifras.

Ejercicio 18

Realiza un programa que nos diga cuántos dígitos tiene un número entero que puede ser positivo o negativo. Se permiten números de hasta 5 dígitos.

Ejercicio 19

Realiza un programa que diga si un número entero positivo introducido por teclado es capicúa. Se permiten números de hasta 5 cifras.

4.3 Repetitivas

Los bucles se utilizan para repetir un conjunto de sentencias.

A. Bucle `for`

Se suele utilizar cuando se conoce previamente el número exacto de iteraciones que se van a realizar. La sintaxis es la siguiente:

```
for (expresion1 ; expresion2 ; expresion3) {  
    sentencias  
}
```

Justo al principio se ejecuta `expresion1`, normalmente se usa para inicializar una variable. El bucle se repite mientras se cumpla `expresion2`. En cada iteración del bucle se ejecuta `expresion3`, que suele ser el incremento o decremento de una variable. A continuación se muestra un ejemplo:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
  </head>  
  <body>  
    <?php  
      for($i = 0 ; $i < 10 ; $i++) {  
        echo "El valor de i es ", $i, "<br>";  
      }  
    ?>  
  </body>  
</html>
```

Bucle `foreach`

Es una variante del bucle `for`, donde no necesitamos controlar el valor de ninguna variable índice para hacer un recorrido por una estructura que contiene más de un valor, tal como ocurre en los arrays. Puesto que no se han estudiado todavía los arrays, veremos esta estructura más adelante.

B. Bucle `while`

El bucle `while` se utiliza para repetir un conjunto de sentencias siempre que se cumpla una determinada condición. Es importante reseñar que la condición se comprueba al comienzo del bucle, por lo que se podría dar el caso de que dicho bucle no se ejecutase nunca. La sintaxis es la siguiente:

```
while (expresion) { sentencias  
  
}
```

Las sentencias se ejecutan una y otra vez mientras `expresion` sea verdadera. El siguiente ejemplo produce la misma salida que el ejemplo anterior, muestra cómo cambian los valores de `$i` del 0 al 9.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $i = 0;
      while( $i < 10 ) {
        echo "El valor de i es ", $i, "<br>";
        $i++;
      }
    ?>
  </body>
</html>
```

Bucle do-while

El bucle `do-while` funciona de la misma manera que el bucle `while`, con la salvedad de que `expresion` se evalúa al final de la iteración. Las sentencias que encierran el bucle `do-while`, por tanto, se ejecutan como mínimo una vez. La sintaxis es la siguiente:

```
do {
  sentencias
} while (expresion)
```

El siguiente ejemplo es el equivalente `do-while` a los dos ejemplos anteriores.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $i = 0;
      do {
        echo "El valor de i es ", $i, "<br>";
        $i++;
      } while( $i < 10 )
    ?>
  </body>
</html>
```

C. Carga reiterada de una página. Adivina el número.

Los bucles `for`, `while` y `do-while` permiten repetir un bloque de sentencias en una carga de página. No obstante, cada vez que un programa necesita pedir información al usuario, es necesario presentar un formulario por pantalla, el usuario rellena ese formulario y los datos son enviados a una página que los procesa que puede ser la misma página que envía los datos. Si se vuelven a necesitar datos del usuario, se repite el proceso. Podemos tener así un bucle, es decir, una repetición de sentencias, que se produce por la carga sucesiva de la misma página y no por la utilización de `for`, `while` o `do-while`

Lo entenderemos mejor con un ejemplo. Vamos a realizar un pequeño juego al que llamaremos “Adivina el número”. El usuario debe adivinar un número entre 0 y 100; el programa irá guiando al usuario diciendo si el número introducido es mayor o menor que el que está pensando el ordenador. Fíjate que si la página se carga por primera vez, lo cual comprobamos preguntando si existe un parámetro, se inicializa `oportunidades` a 5; ese será el número de oportunidades que tendrá el usuario para adivinar el número. La variable `numeroIntroducido` se inicializa de forma arbitraria a 555 simplemente para indicarle a la página principal del juego cuándo se ejecuta por primera vez. Este número se puede cambiar pero debe quedar siempre fuera del intervalo [0 - 100] para que no se confunda con alguno de los números introducidos por el usuario.

A continuación se muestra la página `adivina.php` que constituye el cuerpo del programa:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
    <?php
    if (!isset($_REQUEST['numeroIntroducido'])) {
        $oportunidades = 5;
        $numeroIntroducido=555;
        $numeroSecreto = rand(1,100);
    }else {
        $oportunidades = $_POST['oportunidades'];
        $numeroIntroducido = $_POST['numeroIntroducido'];
        $numeroSecreto = $_POST['numeroSecreto'];
    }
    if ($numeroIntroducido == $numeroSecreto) {
        echo "Enhorabuena, has acertado el número.";
    } else {
        if ($oportunidades == 0) {
            echo "Lo siento, has agotado todas tus oportunidades. Has perdido";
        } else {
```

```

        if ($numeroIntroducido != 555) {
            if ($numeroSecreto > $numeroIntroducido)
                echo "El número que estoy pensando es mayor que el introducido.<br>";
            else
                echo "El número que estoy pensando es menor que el introducido.<br>";
        }
    ?>
    Te quedan <?= $oportunidades-- ?> oportunidades para adivinar el número.<br>
    Introduce un número del 0 al 100
    <form action="adivina.php" method="post">
        <input type="text" name="numeroIntroducido">
        <input type="hidden" name="oportunidades" value="<?= $oportunidades ?>">
        <input type="hidden" name="numeroSecreto" value=" <?= $numeroSecreto ?>">
        <input type="submit" value="Continuar">
    </form>
    <?php
        }
    }
    ?>
</body>
</html>

```

4.4 Ejercicios para practicar

Ejercicio 1

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `for`.

Ejercicio 2

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `while`.

Ejercicio 3

Muestra los números múltiplos de 5 de 0 a 100 utilizando un bucle `do-while`.

Ejercicio 4

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `for`.

Ejercicio 5

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `while`.

Ejercicio 6

Muestra los números del 320 al 160, contando de 20 en 20 utilizando un bucle `do-while`.

Ejercicio 7

Realiza el control de acceso a una caja fuerte. La combinación será un número de 4 cifras. El programa nos pedirá la combinación para abrirla. Si no acertamos, se nos mostrará el mensaje “Lo siento, esa no es la combinación” y si acertamos se nos dirá “La caja fuerte se ha abierto satisfactoriamente”. Tendremos cuatro oportunidades para abrir la caja fuerte.

Ejercicio 8

Muestra la tabla de multiplicar de un número introducido por teclado. El resultado se debe mostrar en una tabla (`<table>` en HTML).

Ejercicio 9

Realiza un programa que nos diga cuántos dígitos tiene un número introducido por teclado.

Ejercicio 10

Escribe un programa que calcule la media de un conjunto de números positivos introducidos por teclado. A priori, el programa no sabe cuántos números se introducirán. El usuario indicará que ha terminado de introducir los datos cuando meta un número negativo.

Ejercicio 11

Escribe un programa que muestre en tres columnas, el cuadrado y el cubo de los 5 primeros números enteros a partir de uno que se introduce por teclado.

Ejercicio 12

Escribe un programa que muestre los n primeros términos de la serie de Fibonacci. El primer término de la serie de Fibonacci es 0, el segundo es 1 y el resto se calcula sumando los dos anteriores, por lo que tendríamos que los términos son 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144... El número n se debe introducir por teclado.

Ejercicio 13

Escribe un programa que lea una lista de diez números y determine cuántos son positivos, y cuántos son negativos.

Ejercicio 14

Escribe un programa que pida una base y un exponente (entero positivo) y que calcule la potencia.

Ejercicio 15

Escribe un programa que dados dos números, uno real (base) y un entero positivo (exponente), saque por pantalla todas las potencias con base el número dado y exponentes entre uno y el exponente introducido. No se deben utilizar funciones de exponenciación. Por ejemplo, si introducimos el 2 y el 5, se deberán mostrar 2^1 , 2^2 , 2^3 , 2^4 , 2^5 .

Ejercicio 16

Escribe un programa que diga si un número introducido por teclado es o no primo. Un número primo es aquel que sólo es divisible entre él mismo y la unidad.

Ejercicio 17

Realiza un programa que sume los 100 números siguientes a un número entero y positivo introducido por teclado. Se debe comprobar que el dato introducido es correcto (que es un número positivo).

Ejercicio 18

Escribe un programa que obtenga los números enteros comprendidos entre dos números introducidos por teclado y validados como distintos, el programa debe empezar por el menor de los enteros introducidos e ir incrementando de 7 en 7.

Ejercicio 19

Realiza un programa que pinte una pirámide por pantalla. La altura se debe pedir por teclado mediante un formulario. La pirámide estará hecha de bolitas, ladrillos o cualquier otra imagen de las 5 que se deben dar a elegir mediante un formulario.

Ejercicio 20

Igual que el ejercicio anterior pero esta vez se debe pintar una pirámide hueca.

Ejercicio 21

Realiza un programa que vaya pidiendo números hasta que se introduzca un número negativo y nos diga cuantos números se han introducido, la media de los impares y el mayor de los pares. El número negativo sólo se utiliza para indicar el final de la introducción de datos pero no se incluye en el cómputo.

Ejercicio 22

Muestra por pantalla todos los números primos entre 2 y 100, ambos incluidos.

Ejercicio 23

Escribe un programa que permita ir introduciendo una serie indeterminada de números hasta que la suma de ellos supere el valor 10000. Cuando esto último ocurra, se debe mostrar el total acumulado, el contador de los números introducidos y la media.

Ejercicio 24

Escribe un programa que lea un número N e imprima una pirámide de números con N filas como en la siguiente figura. Recuerda utilizar un tipo de letra de ancho fijo como por ejemplo Courier para que los espacios tengan la misma anchura que los números. 1

Ejercicio 25

Realiza un programa que pida un número por teclado y que luego muestre ese número al revés.

Ejercicio 26

Realiza un programa que pida primero un número y a continuación un dígito. El programa nos debe dar la posición (o posiciones) contando de izquierda a derecha que ocupa ese dígito en el número introducido.

Ejercicio 27

Escribe un programa que muestre, cuente y sume los múltiplos de 3 que hay entre 1 y un número leído por teclado.

Ejercicio 28

Escribe un programa que calcule el factorial de un número entero leído por teclado.

Ejercicio 29

Escribe un programa que muestre por pantalla todos los números enteros positivos menores a uno leído por teclado que no sean divisibles entre otro también leído de igual forma.

4.5 Conceptos de la unidad

Operadores de comparación

Operador	Nombre	Ejemplo	Devuelve verdadero cuando...
==	Igual	\$a == \$b	\$a es igual \$b (aunque sean de diferente tipo)
===	Igual	\$a === \$b	\$a es igual \$b (y además son del mismo tipo)
!=	Distinto	\$a != \$b	\$a es distinto \$b
<	Menor que	\$a < \$b	\$a es menor que \$b
>	Mayor que	\$a > \$b	\$a es mayor que \$b
<=	Menor o igual	\$a <= \$b	\$a es menor o igual que \$b
>=	Mayor o igual	\$a >= \$b	\$a es mayor o igual que \$b
<=>	Nave espacial	\$a <=> \$b	-1 si \$a es menor 0 si son iguales 1 si \$b es menor

Operadores lógicos

Y -> &&, and	Ejemplos: (7>2) && (2<4), (7>2) and (2<4)
O -> , or	Ejemplos: (7>2) (2<4), (7>2) or (2<4)
Negación -> !	Ejemplos: ! (7>2)

IF Opción 1

```
if (condición) {  
    sentencias a ejecutar cuando la condición es cierta }  
else {  
    Sentencias a ejecutar cuando la condición es falsa }
```


IF Opción 2 (formato resumido)

```
(condición) ? expresión1 : expresión2
```

IF Opción 3

```
<?php if (condition): ?>  
Código html a mostrar si la condición es true  
<?php else: ?>  
Código html a mostrar si la condición es false  
<?php endif ?>
```

SWITCH Opción 1

```
switch(variable) {  
case valor1:  
sentencias  
break;  
case valor2:  
sentencias  
break;  
.  
.  
.  
default:  
sentencias  
}
```

SWITCH Opción 2 (para evaluar expresiones booleanas en vez de un valor)

```
switch(true) {  
case ExpresionBooleana1:  
sentencias  
break;  
case ExpresionBooleana2:  
break;  
.  
.  
.  
default:  
sentencias  
}
```

BUCLE for

```
for (expresion1 ; expresion2 ; expresion3) {  
sentencias  
}
```

BUCLE foreach (para arrays normales)

```
foreach ($array as $elemento) {  
echo $elemento;  
}
```

BUCLE foreach (para arrays asociativos)

```
foreach ($array as $indice=>$valor) {  
echo $elemento;  
}
```

BUCLE while

```
while (expresion) {  
sentencias  
}
```

BUCLE do-while

```
do {  
sentencias  
} while (expresion)
```

4.6 Ejercicios para resolver

Condicionales

Ejercicio 1

Vamos a diseñar un juego para adivinar imágenes mostrando solo alguna parte de ellas. Dividir una imagen en un mosaico de 3x3, y mostrar una cuadrícula en la página principal con todos los cuadrados del mosaico dados la vuelta. Debajo de la cuadrícula habrá una caja de texto para que el usuario intente adivinar el nombre de lo que aparece en la imagen, junto a un botón comprobar.

Cada vez que el usuario pulse en un cuadrado de la cuadrícula se mostrará el contenido solo de esa cuadrícula durante 2 segundos y posteriormente se volverá a ocultar.

Cuando el usuario escriba algo y pulse el botón comprobar ocurrirá lo siguiente:

- Si ha acertado se mostrará la imagen completa y un mensaje de felicitación por acertar.
- Si no ha acertado se mostrará un mensaje indicando que ha fallado y un botón de volver para seguir intentándolo.

Ejercicio 2

Mejora el ejercicio de la lotería primitiva del tema anterior. Ahora los números se seleccionan de un boleto al estilo “bingo” con filas y columnas, para representar los números seleccionados se usarán checkbox, y para el número de serie una caja de texto. Cuando se pulse el botón jugar, se mostrará la combinación ganadora generada aleatoriamente y los aciertos que ha tenido. No se usarán estructuras repetitivas ni arrays, se mostrará la combinación ganadora en una tabla con una sola fila y un número en cada columna.

Se mostrará el número de aciertos que ha tenido el usuario y cuánto dinero ha ganado:

- menos de 4 aciertos: nada
- 4 aciertos: dinero vuelto
- 5 aciertos: 30 euros
- 6 aciertos: 100 euros
- número de serie: Si se acierta se sumarán 500 euros independientemente del número de aciertos.

Nota: no hace falta comprobar todos los números, solo los de la combinación ganadora, por lo que no se controla si el usuario selecciona más de 6 números.

Ejercicio 3 (Opcional)

Partiendo del Ejercicio1, amplíalo con la siguiente funcionalidad. En la pantalla principal se mostrará el número de intentos que lleva el usuario, que en la primera carga será de 0 intentos y conforme se vayan pulsando los cuadrados para descubrir la imagen que esconden se irá aumentando en 1. Cuando el usuario intente adivinar la palabra de la imagen, si acierta se mostrará el número de intentos en que

lo ha conseguido, y si ha fallado mostrará el número de intentos que lleva hasta ese momento, y al volver a la pantalla principal seguirá aumentando el número de intentos que llevaba.

Nota: para controlar el número de intentos es necesario pasarlo como parámetro cada vez que se navega de una página a otra, si no se pierde su valor.

Repetitivas

EJERCICIO 4

Diseñar una página web que muestre una tabla con 3 columnas, la primera corresponde al número de bloque, la segunda al piso y en la tercera hay un botón llamar. En total hay 10 bloques y cada bloque tiene 7 pisos. Cuando se pulse llamar en un piso de un bloque, mostrará en otra página el mensaje del tipo “Usted ha llamado al piso 3 del bloque 6”. Utiliza estructuras repetitivas para generar la tabla.

EJERCICIO 5

Diseñar una página que esté compuesta por una tabla de 10 filas por 10 columnas, y en cada celda habrá una imagen de un ojo cerrado. Cada vez que el usuario pulse un ojo, se recargará la página con todos los ojos cerrados salvo el que se ha pulsado que corresponderá a un ojo abierto.

EJERCICIO 6

Realiza el ejercicio 1 correspondiente al juego de adivinar una imagen, pero usando estructuras repetitivas para simplificar el código.

EJERCICIO 7

Realiza el ejercicio 2 correspondiente al juego de la primitiva, pero usando estructuras repetitivas para simplificar el código. Pero en esta ocasión lo vamos a realizar de una forma más vistosa gracias a que podemos utilizar las estructuras repetitivas. Para mostrar el resultado de la apuesta vamos a mostrar una tabla con todos los números de la primitiva, y los números elegidos por el usuario que hayan sido aciertos serán de color verde, los elegidos por el usuario que no han sido aciertos serán de color negro, los números de la combinación aleatoria que no han sido elegidos por el usuario serán de color rojo y por último el resto de números serán de color gris. También contaremos los números seleccionados por el usuario y si son más de 6, en vez de mostrar el premio obtenido se mostrará un mensaje indicando que ha hecho trampas.

5. Arrays

5.1 Arrays clásicos

Un array es un tipo de dato capaz de almacenar múltiples valores. Se utiliza cuando tenemos muchos datos parecidos, por ejemplo, para almacenar la temperatura media diaria en Málaga durante el último año podríamos utilizar las variables \$temp0, \$temp1, \$temp2, \$temp3, \$temp4, ... y así hasta 365 variables distintas, pero sería poco práctico; es mejor utilizar un array de nombre \$temp y usar un índice para referirnos a una temperatura concreta.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $temp[0] = 16;
      $temp[1] = 15;
      $temp[2] = 17;
      $temp[3] = 15;
      $temp[4] = 16;
      echo "La temperatura en Málaga el cuarto día del año fue de "; echo $temp[3],
      "°C";
    ?>
  </body>
</html>
```

Los valores de un array se pueden asignar directamente en una línea. El índice comienza en 0.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $temp = array(16, 15, 17, 15, 16);
      echo "La temperatura en Málaga el cuarto día del año fue de ";
      echo $temp[3], "°C";
    ?>
  </body>
</html>
```

En este otro ejemplo, asignamos valores aleatorios a los elementos de un array.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      echo "<b>Notas:</b><br>";
      for ($i = 0; $i < 10; $i++) {
        // números aleatorios entre 0 y 10 (ambos incluidos)
        $n[$i] = rand(0, 10);
      }
      foreach ($n as $elemento) {
        echo $elemento, "<br>";
      }
    ?>
  </body>
</html>
```

A partir de PHP 5.4 se puede utilizar la sintaxis abreviada (utilizando corchetes) para definir un array.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $color = ["verde", "amarillo", "rojo", "azul", "blanco", "gris"];
      echo "Mañana me pongo una camiseta de color ", $color[rand(0, 5)], ".";
    ?>
  </body>
</html>
```

Como puedes ver en los ejemplos anteriores, no es necesario definir previamente un array antes de utilizarlo. Tampoco hay límite en cuanto al número de elementos que se pueden añadir al mismo. No obstante, se pueden crear arrays de tamaño fijo con `SplFixedArray` que son más eficientes en cuanto a uso de la memoria y más rápidas en las operaciones de lectura y escritura.

La función `var_dump()` se utiliza para mostrar el tipo y el valor de un dato, en este caso muestra los tipos y valores de cada uno de los elementos del array.

```
<!DOCTYPE html>
<html>
  <head>
```

```

    <meta charset="UTF-8">
</head>
<body>
    <?php
        $a = new SplFixedArray(10);
        $a[0] = 843;
        $a[2] = 11;
        $a[6] = 1372;
        // Los valores del array que no se han inicializado son NULL
        foreach ($a as $elemento) {
            var_dump($elemento);
            echo "<br>";
        }
    ?>
</body>
</html>

```

5.2 Arrays asociativos

En un array asociativo se pueden utilizar índices que no son numéricos, a modo de claves. Veamos un ejemplo de un array asociativo que almacena alturas (en centímetros) y que como índice o clave utiliza nombres.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head> <body>
        <?php
            $estatura = array("Rosa"=>168, "Ignacio"=>175, "Daniel"=>172, "Rubén"=>182);
            echo "La estatura de Daniel es ", $estatura['Daniel'], " cm"; ?>
        </body>
</html>

```

También se pueden asignar los valores de forma individual:

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <?php
            $estatura['Rosa'] = 168;
            $estatura['Ignacio'] = 175;
            $estatura['Daniel'] = 172;

```

```

    $estatura['Rubén'] = 182;
    echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm"; ?>
</body>
</html>

```

Si el lector es suficientemente perspicaz ya se habrá dado cuenta de que \$_GET y \$_POST son arrays asociativos.

Con los arrays asociativos se puede usar también la sintaxis abreviada que emplea corchetes.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $estatura = ["Rosa" => 168, "Ignacio" => 175, "Daniel" => 172, "Rubén" => 182];
      echo "La estatura de Daniel es ", $estatura['Daniel'] , " cm"; ?>
    </body>
  </html>

```

5.3 Arrays bidimensionales

Un array bidimensional utiliza dos índices para localizar cada elemento. Podemos ver este tipo de datos como un array que, a su vez, contiene otros arrays. En el siguiente ejemplo se define un array con 4 elementos que, a su vez, son un array asociativo cada uno de ellos.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $persona = array (
        array( "nombre" => "Rosa", "estatura" => 168, "sexo" => "F"),
        array( "nombre" => "Ignacio", "estatura" => 175, "sexo" => "M"),
        array( "nombre" => "Daniel", "estatura" => 172, "sexo" => "M"),
        array( "nombre" => "Rubén", "estatura" => 182, "sexo" => "M"));
      echo "<b>DATOS SOBRE EL PERSONAL<b><br><hr>";
      $numPersonas = count($persona);
      for ($i = 0; $i < $numPersonas; $i++) {
        echo "Nombre: <b>", $persona[$i]['nombre'], "</b><br>";
        echo "Estatura: <b>", $persona[$i]['estatura'], " cm</b><br>";
      }
    </body>
  </html>

```

```

        echo "Sexo: <b>", $persona[$i]['sexo'], "</b><br><hr>";
    }
?>
</body>
</html>

```

Observa que la función `count()` permite saber el número de elementos de un array.

5.4 Iterador `foreach`

El iterador `foreach` se utiliza para recorrer todos los elementos de un array sin que tengamos que preocuparnos por los índices ni por el tamaño del array.

Es común cometer errores al utilizar arrays por no establecer bien el valor inicial o el valor final en el bucle que la recorre, o por no determinar bien el tamaño. Con `foreach` nos podemos despreocupar de todo eso, simplemente recorremos todo el array de principio a fin. Veamos un ejemplo.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    <?php
      $cajonDeSastre = [30, -7, "Me gusta el queso", 56, "¡eh!", 237];
      foreach ($cajonDeSastre as $cosa) {
        echo "$cosa<br>";
      }
    ?>
  </body>
</html>

```

5.5 Cómo recoger datos para un array mediante un formulario

Imagina que quieres pedir diez números por teclado y quieres guardar esos números en un array. Se puede pedir un número mediante un formulario, a continuación, el siguiente, luego el siguiente, etc. pero ¿cómo enviarlos? Hay un truco muy sencillo. Se pueden ir concatenando valores en una cadena de caracteres y el resultado de esa concatenación se puede reenviar una y otra vez en un formulario como campo oculto. Por último, se puede utilizar la función `explode()` para convertir la cadena de caracteres en un array.

Es importante señalar que los valores que se van concatenando deben tener algún tipo de separación dentro de la cadena, por ejemplo un espacio en blanco.

A continuación, se muestra un ejemplo completo:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
</head>
<body>
    <?php
    if (isset($_POST['n'])) {
        $contadorNumeros = $_POST['contadorNumeros'];
        $numeroTexto = $_POST['numeroTexto'] . " ". $_POST['n'];
    } else {
        $contadorNumeros = 0;
        $numeroTexto = "";
    }
    // Muestra los números introducidos
    if ($contadorNumeros == 6) {
        $numeroTexto = substr($numeroTexto, 1); // quita el primer espacio
        $numero = explode(" ", $numeroTexto);
        echo "Los números introducidos son: ";
        foreach ($numero as $n) {
            echo $n, " ";
        }
    } else {
    ?>
        <form action="#" method="post">
        Introduzca un número:
        <input type="number" name="n" autofocus>
        <input type="hidden" name="contadorNumeros" value="<?= ++$contadorNumeros ?>">
        <input type="hidden" name="numeroTexto" value="<?= $numeroTexto ?>">
        <input type="submit" value="OK">
        </form>
    <?php
    }
    ?>
</body>
</html>
```

5.6 Ejercicios para practicar

Ejercicio 1

Define tres arrays de 20 números enteros cada una, con nombres “numero”, “cuadrado” y “cubo”. Carga el array “numero” con valores aleatorios entre 0 y 100. En el array “cuadrado” se deben almacenar los cuadrados de los valores que hay en el array “numero”. En el array “cubo” se deben almacenar los cubos

de los valores que hay en “numero”. A continuación, muestra el contenido de los tres arrays dispuesto en tres columnas.

Ejercicio 2

Escribe un programa que pida 10 números por teclado y que luego muestre los números introducidos junto con las palabras “máximo” y “mínimo” al lado del máximo y del mínimo respectivamente.

Ejercicio 3

Escribe un programa que lea 15 números por teclado y que los almacene en un array. Rota los elementos de ese array, es decir, el elemento de la posición 0 debe pasar a la posición 1, el de la 1 a la 2, etc. El número que se encuentra en la última posición debe pasar a la posición 0. Finalmente, muestra el contenido del array.

Ejercicio 4

Escribe un programa que genere 100 números aleatorios del 0 al 20 y que los muestre por pantalla separados por espacios. El programa pedirá entonces por teclado dos valores y a continuación cambiará todas las ocurrencias del primer valor por el segundo en la lista generada anteriormente. Los números que se han cambiado deben aparecer resaltados de un color diferente.

Ejercicio 5

Realiza un programa que pida la temperatura media que ha hecho en cada mes de un determinado año y que muestre a continuación un diagrama de barras horizontales con esos datos. Las barras del diagrama se pueden dibujar a base de la concatenación de una imagen.

Ejercicio 6

Realiza un programa que pida 8 números enteros y que luego muestre esos números de colores, los pares de un color y los impares de otro.

Ejercicio 7

Escribe un programa que genere 20 números enteros aleatorios entre 0 y 100 y que los almacene en un array. El programa debe ser capaz de pasar todos los números pares a las primeras posiciones del array (del 0 en adelante) y todos los números impares a las celdas restantes. Utiliza arrays auxiliares si es necesario.

Ejercicio 8

Realiza un programa que pida 10 números por teclado y que los almacene en un array. A continuación se mostrará el contenido de ese array junto al índice (0 – 9) utilizando para ello una tabla. Seguidamente el

programa pasará los primos a las primeras posiciones, desplazando el resto de números (los que no son primos) de tal forma que no se pierda ninguno. Al final se debe mostrar el array resultante.

Por ejemplo:

Array inicial									
0	1	2	3	4	5	6	7	8	9
20	5	7	4	32	9	2	14	11	6

Array final									
0	1	2	3	4	5	6	7	8	9
5	7	2	11	20	4	32	9	14	6

Ejercicio 9

Realiza un programa que pida 10 números por teclado y que los almacene en un array. A continuación se mostrará el contenido de ese array junto al índice (0 – 9). Seguidamente el programa pedirá dos posiciones a las que llamaremos “inicial” y “final”. Se debe comprobar que inicial es menor que final y que ambos números están entre 0 y 9. El programa deberá colocar el número de la posición inicial en la posición final, rotando el resto de números para que no se pierda ninguno.

Al final se debe mostrar el array resultante.

Por ejemplo:

Array inicial									
0	1	2	3	4	5	6	7	8	9
20	5	7	4	32	9	2	14	11	6

Posición inicial: 3

Posición final: 7

Array final									
0	1	2	3	4	5	6	7	8	9
6	20	5	7	32	9	2	4	14	11

Ejercicio 10

Realiza un programa que escoja al azar 10 cartas de la baraja española y que diga cuántos puntos suman según el juego de la brisca. Emplea un array asociativo para obtener los puntos a partir del nombre de la figura de la carta. Asegúrate de que no se repite ninguna carta, igual que si las hubieras cogido de una baraja de verdad.

Ejercicio 11

Crea un mini-diccionario español-inglés que contenga, al menos, 20 palabras (con su traducción). Utiliza un array asociativo para almacenar las parejas de palabras. El programa pedirá una palabra en español y dará la correspondiente traducción en inglés.

Ejercicio 12

Realiza un programa que escoja al azar 5 palabras en español del mini-diccionario del ejercicio anterior. El programa pedirá que el usuario teclee la traducción al inglés de cada una de las palabras y comprobará si son correctas. Al final, el programa deberá mostrar cuántas respuestas son válidas y cuántas erróneas.

Ejercicio 13

Rellena un array bidimensional de 6 filas por 9 columnas con números enteros positivos comprendidos entre 100 y 999 (ambos incluidos). Todos los números deben ser distintos, es decir, no se puede repetir ninguno. Muestra a continuación por pantalla el contenido del array de tal forma que se cumplan los siguientes requisitos:

- Los números de las dos diagonales donde está el mínimo deben aparecer en color verde.
- El mínimo debe aparecer en color azul.
- El resto de números debe aparecer en color negro.

Ejercicio 14

Escribe un programa que, dada una posición en un tablero de ajedrez, nos diga a qué casillas podría saltar un alfil que se encuentra en esa posición. Indícalo de forma gráfica sobre el tablero con un color diferente para estas casillas donde puede saltar la figura. El alfil se mueve siempre en diagonal. El tablero cuenta con 64 casillas. Las columnas se indican con las letras de la “a” a la “h” y las filas se indican del 1 al 8.

Ejercicio 15

Realiza un programa que sea capaz de rotar todos los elementos de una matriz cuadrada una posición en el sentido de las agujas del reloj. La matriz debe tener 12 filas por 12 columnas y debe contener números generados al azar entre 0 y 100. Se debe mostrar tanto la matriz original como la matriz resultado, ambas con los números convenientemente alineados.

5.7 Conceptos de la unidad

Arrays clásicos de tamaño variable

```
$v[0] = 16; $v[1] = 15; $v[2] = 17; $v[3] = 15; $v[4] = 16;  
$v = array(16, 15, 17, 15, 16);  
$color = ["verde", "amarillo", "rojo", "azul", "blanco", "gris"];  
$v[]=14; Añade el valor en la última posición  
echo $v[4]; acceso al valor del índice 4 del array  
$v[]="nuevo valor"; //añade un valor al final del array
```

Como paso un array como parámetro a otra página

Solo se puede pasar texto, así que hay que convertir el array en una cadena antes de enviarlo y obtener el array a partir de la cadena en la página de destino.

Opción 1:

```
$v = explode("caracter", $texto); separa el texto por cada 'carácter' encontrado y lo almacena en el array  
$texto = implode("caracter", $v); une los elementos del array en una cadena, con el 'carácter' indicado entre los elementos del array
```

Opción 2:

```
$cadena= serialize($array); //$cadena almacena el array como texto (para usar en BD o sesiones)  
$array= unserialize($cadena); //se recupera el array u objeto original serializado en $cadena
```

Nota: con arrays asociativos la serialización se corrompe al ser enviada, así que hay que usar la función en combinación con otra de la siguiente forma:

```
$cadena=base64_encode(serialize($array));  
$array=unserialize(base64_decode($cadena));
```

Arrays clásicos de tamaño fijo:

```
$v = new SplFixedArray(10); (los valores no inicializados son null)
```

Arrays asociativos:

```
$edades['Rosa']=16; $edades['Ignacio']=25; $edades['Daniel']=17; $edades['Rubén']=18;  
$edades = array("Rosa" => 16, "Ignacio" => 25, "Daniel" => 17, "Rubén" => 18);  
$edades = ["Rosa" => 16, "Ignacio" => 25, "Daniel" => 17, "Rubén" => 18];  
echo "Daniel: ", $edades['Daniel'];
```

Paso en un formulario de varios valores como un array

En el formulario:

```
<input type="text" name="nombreArray[indice1]">  
<input type="text" name="nombreArray[indice2]">
```

...

En página de destino.

```
$nombreArray=$_GET['nombreArray'];
```

Arrays bidimensionales:

```
$v = array(array(5, 6, 2), array(4, 7, 1, 6, 3), array(5, 9));  
$v = [[5, 6, 2], [4, 7, 1, 6, 3], [5, 9]];  
$persona = array (  
    array( "nombre" =>"Rosa", "estatura" => 168, "sexo" =>"F"),  
    array( "nombre" =>"Ignacio", "estatura" => 175, "sexo" =>"M"),  
    array( "nombre" =>"Daniel", "estatura" => 172, "sexo" =>"M"),  
    array( "nombre" =>"Rubén", "estatura" => 182, "sexo" =>"M")  
);  
$persona = [['ana'=>'pepe', 'julia'=>'juan'],  
    ['luisa'=>'adrian'],  
    ['eva'=>'wally', 'sandra'=>'antonio','maria'=>'jose']];  
$persona = ['ancianos'=> ['ana'=>'pepe', 'julia'=>'juan'], 'mayores'=> ['luisa'=>'adrian'],  
    'jovenes'=> ['eva'=>'wally', 'sandra'=>'antonio','maria'=>'jose']];
```

Funciones principales de Arrays:

count(\$array): devuelve la dimensión de un array

array_key_last(\$array): devuelve el índice del último elemento añadido al array asociativo o clásico

in_array(\$buscado, \$array): devuelve booleano si el elemento buscado está en array

array_key_exists(\$buscado,\$array): devuelve booleano si elemento buscado en índices del array asociativo

array_search(\$buscado, \$array): devuelve el índice (si es asociativo será una cadena) del elemento buscado en array, si no está devuelve falso.

array_rand(\$array, \$num): devuelve un array de tantos elementos como indique \$num, con valores aleatorios elegidos de entre las claves/índices de \$array (sea asociativo o no)

array_fill(índice_comienzo, número_elementos, valor): crea e inicializa un array, comenzando en índice_comienzo, con número_elementos y con el valor indicado en todas las posiciones.

array_slice(\$array, posición_inicio, número_elementos): devuelve del array, solo con los elementos indicados: desde la posición de inicio (si es negativo se cuenta desde el final hacia atrás) y la cantidad indicada en número de elementos (si no se indica, se coge hasta el último).

Recorrido arrays con foreach

```
//para arrays normales  
foreach ($array as $elemento) {  
    echo $elemento;
```

```

}

//para arrays asociativos
foreach ($array as $indice=>$valor) {
    echo $valor;
}

```

Recorrido array bidimensional clásico

```

for($i=0; $i<count($m); $i++){
    for ($j=0; $j<count($m[$i]); $j++){
        echo $m[$i][$j], ' - ';
    }
    echo '<br>';
}

```

Recorrido array bidimensional asociativo (en 1 dimensión)

Asociativo en la 1ª dimensión

```

foreach ($alumno as $nombre=>$notas){
    $suma=0;
    for ($i=0; $i<count($notas); $i++){
        $suma+=$notas[$i];
    }
    $media=$suma/count($notas);
    echo "Alumno: $nombre – Nota media: $media <br>";
}

```

Asociativo en la 2ª dimensión

```

for ($i=0; $i<count($persona); $i++){
    foreach ($persona[$i] as $m=>$h){
        echo $m, ' casada con ', $h, ' ';
    }
    echo '<br>';
}

```

Recorrido array bidimensional asociativo (en las 2 dimensiones)

```

foreach($persona as $ciudad=>$filaPersonas){
    echo "Parejas de la ciudad $ciudad ";
    foreach ($filaPersonas as $m=>$h){
        echo $m, ' casada con ', $h, ' ';
    }
}

```

```
    echo '<br>';  
}
```


5.8 Ejercicios para resolver

Ejercicio 1

Diseñar una página que esté compuesta por una tabla de 10 filas por 10 columnas, y en cada celda habrá una imagen de un ojo cerrado. Cada vez que el usuario pulse un ojo, se recargará la página con todos los ojos cerrados salvo los que se han ido pulsando que corresponderá a un ojo abierto. Conforme se vayan pulsando más ojos se irá completando la tabla de ojos abiertos. Si se pulsa en un ojo abierto se volverá a cerrar. Usar la función `explode()` para pasar arrays como cadenas.

Ejercicio 2

Realizar una página web para realizar pedidos de comida rápida. Tenemos varios tipos de comida: Pizza: jamon, atun, bacon, pepperoni; Hamburguesa: lechuga, tomate, queso; Perrito caliente: lechuga, cebolla, patata; etc (la carta con todas las comidas y sus ingredientes estará almacenada en un array). A través de formularios distintos, uno para cada tipo de comida se va añadiendo comida al pedido con sus respectivos ingredientes, hasta que se pulse el botón finalizar pedido (en otro formulario), con lo que se mostrará el pedido completo en una tabla, con toda la comida y los ingredientes seleccionados de cada una. Usa la función `serialize()` y `unserialize()` para pasar arrays como cadenas.

Ejercicio 3

Vamos a hacer el ejercicio de adivinar la imagen oculta del tema 3 más interesante. Para empezar, vamos a hacer el mosaico un poco más grande, de 10x10. Además la imagen no se va a dividir sino que formará parte del fondo de la tabla y para ocultar o mostrar cada parte del mosaico, el fondo de cada celda será transparente u opaco. Cada vez que se pulse un cuadrado, la parte de la imagen correspondiente a ese cuadrado se mostrará de manera definitiva, así que cada vez se irán mostrando más partes de la imagen. Por último, para rematar y hacerlo todavía más interesante, vamos a poner un límite en el número de cuadrados volteados, de modo que, si no se adivina la imagen antes de superar ese límite, se mostrará un mensaje indicando que ha perdido junto a la imagen completa. Si se acierta introduciendo el nombre correcto en la caja de texto antes de superar el límite, también se indicará con un mensaje junto a la imagen completa. Ayuda: La tabla con las partes visibles y no visibles de la imagen, se encuentra en una única página que se recarga con cada pulsación, pero el resultado del juego tanto si ha ganado como si ha perdido, se puede realizar en otra página distinta. Almacenar en un array la situación de cada celda (vista u oculta) y pasar el array con la función `serialize()` para mayor comodidad.

Ejercicio 4

Vamos a realizar una página para generar parejas compatibles según su sexo y orientación sexual. Para ello en una primera página pediremos de manera reiterada el nombre, sexo (h o m) y orientación (heterosexual, homosexual o bisexual) de personas, que se irán almacenando en un array. Se dispondrá además, de un botón para pasar a la segunda página que permite generar las parejas con las personas introducidas.

En esta segunda página se mostrará un listado en una tabla con todas las personas introducidas y en la última columna debe haber un botón para seleccionar la persona, lo que provocará la recarga de la página, quedando la persona seleccionada de un color distinto y además tras seleccionar a la persona, junto a la primera tabla debe aparecer otra tabla con las personas compatibles con la persona seleccionada. En esta segunda tabla de personas compatibles, en la última columna debe haber un botón para formar la pareja (la primera persona elegida junto con la seleccionada de la segunda tabla) y se enviarán las dos personas seleccionadas a una tercera página que mostrará la información de la pareja formada.

Para las compatibilidades se tendrá en cuenta lo siguiente:

- Una persona heterosexual será compatible con alguien de sexo distinto no homosexual.
- Una persona homosexual será compatible con alguien de mismo sexo no heterosexual
- Una persona bisexual será compatible con cualquiera menos heterosexual del mismo sexo u homosexual de sexo distinto.

Nota: El array de personas puede tener la siguiente estructura (se recomienda inicializar el array con datos para mayor comodidad en las pruebas):

```
$personas= [['nombre'=>'Anita', 'sexo'=>'m', 'orientacion'=>'bis'],  
            ['nombre'=>'Lolita', 'sexo'=>'m', 'orientacion'=>'bis'],  
            ['nombre'=>'Pepito', 'sexo'=>'h', 'orientacion'=>'bis'],  
            ['nombre'=>'Juanito', 'sexo'=>'h', 'orientacion'=>'bis'],  
            ['nombre'=>'Roberto', 'sexo'=>'h', 'orientacion'=>'het'],  
            ['nombre'=>'Antonio', 'sexo'=>'h', 'orientacion'=>'het'],  
            ['nombre'=>'Manuela', 'sexo'=>'m', 'orientacion'=>'het'],  
            ['nombre'=>'Isabel', 'sexo'=>'m', 'orientacion'=>'het'],  
            ['nombre'=>'Jenifer', 'sexo'=>'m', 'orientacion'=>'hom'],  
            ['nombre'=>'Susan', 'sexo'=>'m', 'orientacion'=>'hom'],  
            ['nombre'=>'Peter', 'sexo'=>'h', 'orientacion'=>'hom'],  
            ['nombre'=>'Mike', 'sexo'=>'h', 'orientacion'=>'hom']];
```

Ejercicio 5

Realizar una aplicación web que permita introducir los datos de unos aspirantes a un trabajo. Para ello en la página principal se deberá mostrar un formulario para introducir los siguientes datos: Nombre, edad, años de experiencia y correo. Tendremos un botón para aceptar los datos introducidos del aspirante y otro para finalizar la lista de aspirantes. La aplicación deberá ir almacenando los datos de los aspirantes en un array asociativo, cuyo índice principal sea el nombre. Cuando se pulse el botón Finalizar, se enviarán los datos a otra página para mostrar el listado de los aspirantes, y como se buscan un perfil joven, los mayores de 30 saldrán con el texto de color rojo.

Si se carga la segunda página sin enviar el listado desde la primera, se mostrará un mensaje indicando que primero se deben introducir los aspirantes y un enlace a la primera página.

6. Funciones

6.1 Implementando funciones para reutilizar código

En programación es muy frecuente reutilizar código, es decir, usar código ya existente. Cuando una parte de un programa requiere una funcionalidad que ya está implementada en otro programa no tiene mucho sentido emplear tiempo y energía en implementarla otra vez.

Una función es un trozo de código que realiza una tarea muy concreta y que se puede incluir en cualquier programa cuando hace falta resolver esa tarea. Opcionalmente, las funciones aceptan una entrada (parámetros de entrada) y devuelven una salida.

Observa el siguiente ejemplo. Se trata de un programa que pide un número mediante un formulario y luego dice si el número introducido es o no es primo.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      if (!isset($_POST['n'])) {
        ?>
        Introduzca un número para saber si es primo o no.<br>
        <form action=numeroPrimo.php method="post">
          <input type="number" name="n" min="0" autofocus><br>
          <input type="submit" value="Aceptar">
        </form>
        <?php
        } else {
          $n = $_POST['n'];
          $esPrimo = true;
          for ($i = 2; $i < $n; $i++) {
            if ($n % $i == 0) {
              $esPrimo = false;
            }
          }
          // El 0 y el 1 no se consideran primos
          if (($n == 0) || ($n == 1)) { $esPrimo = false;
          }
          if ($esPrimo) { echo "El $n es primo.";
          } else { echo "El $n no es primo.";
          }
        }
      }
    ?>
```

```

    </body>
</html>

```

Podemos intuir que la tarea de averiguar si un número es o no primo será algo que utilizaremos con frecuencia más adelante así que podemos aislar el trozo de código que realiza ese cometido para usarlo con comodidad en otros programas.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // Programa principal //////////////////////////////////////
      if (!isset($_POST['numero'])) {
        ?>
        Introduzca un número para saber si es primo o no.<br>
        <form action=numeroPrimoConFuncion.php method="post">
          <input type="number" name="numero" min="0" autofocus><br>
          <input type="submit" value="Aceptar">
        </form>
        <?php
        } else {
          $numero = $_POST['numero'];
          if (esPrimo($numero)) {
            echo "El $numero es primo.";
          } else {
            echo "El $numero no es primo.";
          }
        }
      }
      // Funciones //////////////////////////////////////
      function esPrimo($n) {
        for ($i = 2; $i < $n; $i++) {
          if ($n % $i == 0) {
            return false;
          }
        }
        // El 0 y el 1 no se consideran primos
        if (($n == 0) || ($n == 1)) {
          return false;
        }
        return true;
      }
    ?>
  </body>
</html>

```

Cada función tiene una cabecera y un cuerpo. En el ejemplo anterior la cabecera es

```
function esPrimo($n)
```

La función `esPrimo()` toma como parámetro un número entero y devuelve un valor lógico (`true` o `false`). Observa que, a diferencia de otros lenguajes de programación fuertemente tipados como Java, en PHP no hace falta indicar el tipo de dato que devuelve la función ni el/los tipo/s de datos de los parámetros que se pasan.

6.2 Creación de bibliotecas de funciones

Por lo general y salvo casos puntuales, las funciones se suelen agrupar en ficheros. Estos ficheros de funciones se incluyen posteriormente en el programa principal mediante `include` o `include_once` seguido del nombre completo del fichero.

Veamos cómo utilizar la función `esPrimo()` desde un fichero independiente.

El siguiente código corresponde al fichero `matematicas.php`.

```
<?php
function esPrimo($n) {
    $esPrimo = true;
    for ($i = 2; $i < $n; $i++) {
        if ($n % $i == 0) {
            $esPrimo = false;
        }
    }
    // El 0 y el 1 no se consideran primos
    if (($n == 0) || ($n == 1)) {
        $esPrimo = false;
    }
    return $esPrimo;
}
```

El programa principal en otro archivo php, es el siguiente.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <?php
        // Carga las funciones matemáticas
        include 'matematicas.php';
        if (!isset($_POST['numero'])) {
            ?>
            Introduzca un número para saber si es primo o no.<br>
            <form action=numeroPrimo2.php method="post">
                <input type="number" name="numero" min="0" autofocus><br>
```

```
        <input type="submit" value="Aceptar">
    </form>
    <?php
    } else {
        $numero = $_POST['numero'];
        if (esPrimo($numero)) {
            echo "El $numero es primo.";
        } else {
            echo "El $numero no es primo.";
        }
    }
    ?>
</body>
</html>
```

6.4 Ejercicios para practicar

Ejercicios 1-14

Crea una biblioteca de funciones matemáticas que contenga las siguientes funciones. Recuerda que puedes usar unas dentro de otras si es necesario.

1. `esCapicua`: Devuelve verdadero si el número que se pasa como parámetro es capicúa y falso en caso contrario.
2. `esPrimo`: Devuelve verdadero si el número que se pasa como parámetro es primo y falso en caso contrario.
3. `siguientePrimo`: Devuelve el menor primo que es mayor al número que se pasa como parámetro.
4. `potencia`: Dada una base y un exponente devuelve la potencia.
5. `digitos`: Cuenta el número de dígitos de un número entero.
6. `voltea`: Le da la vuelta a un número.
7. `digitoN`: Devuelve el dígito que está en la posición `n` de un número entero. Se empieza contando por el 0 y de izquierda a derecha.
8. `posicionDeDigito`: Da la posición de la primera ocurrencia de un dígito dentro de un número entero. Si no se encuentra, devuelve -1.
9. `quitaPorDetras`: Le quita a un número `n` dígitos por detrás (por la derecha).
10. `quitaPorDelante`: Le quita a un número `n` dígitos por delante (por la izquierda).
11. `pegaPorDetras`: Añade un dígito a un número por detrás.
12. `pegaPorDelante`: Añade un dígito a un número por delante.
13. `trozoDeNumero`: Toma como parámetros las posiciones inicial y final dentro de un número y devuelve el trozo correspondiente.
14. `juntaNumeros`: Pega dos números para formar uno.

Ejercicio 15

Muestra los números primos que hay entre 1 y 1000.

Ejercicio 16

Muestra los números capicúa que hay entre 1 y 99999.

Ejercicio 17

Escribe un programa que pase de binario a decimal.

Ejercicio 18

Escribe un programa que pase de decimal a binario.

Ejercicio 19

Une y amplía los dos programas anteriores de tal forma que se permita convertir un número entre cualquiera de las siguientes bases: decimal, binario, hexadecimal y octal.

Ejercicios 20-28

Crea una biblioteca de funciones para arrays (de una dimensión) de números enteros que contenga las siguientes funciones:

1. `generaArrayInt`: Genera un array de tamaño `n` con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
2. `minimoArrayInt`: Devuelve el mínimo del array que se pasa como parámetro.
3. `maximoArrayInt`: Devuelve el máximo del array que se pasa como parámetro.
4. `mediaArrayInt`: Devuelve la media del array que se pasa como parámetro.
5. `estaEnArrayInt`: Dice si un número está o no dentro de un array.
6. `posicionEnArray`: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra.
7. `volteaArrayInt`: Le da la vuelta a un array.
8. `rotaDerechaArrayInt`: Rota `n` posiciones a la derecha los números de un array.
9. `rotaIzquierdaArrayInt`: Rota `n` posiciones a la izquierda los números de un array.

Ejercicio 29-34

Crea una biblioteca de funciones para arrays bidimensionales (de dos dimensiones) de números enteros que contenga las siguientes funciones:

1. `generaArrayBiInt`: Genera un array de tamaño `n x m` con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
2. `filaDeArrayBiInt`: Devuelve la fila `i`-ésima del array que se pasa como parámetro.
3. `columnaDeArrayBiInt`: Devuelve la columna `j`-ésima del array que se pasa como parámetro.
4. `coordenadasEnArrayBiInt`: Devuelve la fila y la columna (en un array con dos elementos) de la primera ocurrencia de un número dentro de un array bidimensional. Si el número no se encuentra en el array, la función devuelve el array `{-1, -1}`.
5. `esPuntoDeSilla`: Dice si un número es o no punto de silla, es decir, mínimo en su fila y máximo en su columna.
6. `diagonal`: Devuelve un array que contiene una de las diagonales del array bidimensional que se pasa como parámetro. Se pasan como parámetros fila, columna y dirección. La fila y la columna determinan el número que marcará las dos posibles diagonales dentro del array. La dirección es una cadena de caracteres que puede ser "nose" o "neso". La cadena "nose" indica que se elige la diagonal que va del noroeste hacia el sureste, mientras que la cadena "neso" indica que se elige la diagonal que va del noreste hacia el suroeste.

6.5 Ejercicios para resolver

Ejercicio 1.

Crear una página web para generar de manera aleatoria una combinación de apuesta en la lotería primitiva. Se pedirán 6 números (entre 1 y 49) y el número de serie (entre 1 y 999). El usuario podrá rellenar los números pedidos que desee, dejando en blanco el resto, de modo que la combinación generada respete los números elegidos y genere aleatoriamente el resto hasta completar la combinación (el número de serie también es opcional). El usuario también podrá rellenar de manera opcional una caja de texto como título para su combinación.

Crear una función `combinacion()` que sea llamada para generar la combinación en función de los parámetros recibidos y devuelva el array generado.

Crear una función `imprimeApuesta()` que reciba un array y un texto, y devuelva una cadena con el código hml de una tabla con 2 filas, la primera fila con el texto y la segunda con el array recibido. Usaremos esta función para mostrar el resultado de la combinación generada. Si la función no recibe ningún texto como título en la primera fila incluirá el texto "Combinación generada para la Primitiva".

Ejercicio 2.

Disponemos de 2 tarjetas de coordenadas para controlar el acceso a una web. Cada tarjeta corresponde a un perfil de usuario 'admin' o 'estandar', cada número registrado en la tarjeta se identifica por su fila (de la 1 a la 5) y su columna (de la A a la E). Los valores registrados en cada tarjeta son fijos y os lo podéis inventar. Crea una página principal que sirva de control de acceso a una segunda página. Se pedirá el perfil de usuario (admin o estándar) y una clave aleatoria correspondiente a la tarjeta de coordenadas de su perfil (fila y columna), se comprobará si es correcto usando 2 funciones: dameTarjeta() a la que se le pasa el perfil y devuelve una matriz correspondiente a la tarjeta de coordenadas de ese perfil, y compruebaClave() a la que se le pasa la matriz de coordenadas, las coordenadas y un valor, y devolverá un booleano según sea correcto el valor en la matriz de coordenadas. Ambas funciones estarán almacenadas en una librería controlAcceso.php. Si el usuario se ha identificado correctamente se muestra un enlace de acceso a la página protegida (cualquiera) y si no mostrará un enlace para volver a intentarlo de nuevo.

Usar una tercera función imprimeTarjeta() que recibe una tarjeta y devuelve el código html correspondiente a una tabla con el el valor de todas las coordenadas. (imprimir las tarjetas de cada perfil en la página de acceso para poder comprobar el correcto funcionamiento de la página)

6.6 Conceptos de la unidad

```
function nombreFuncion ($par1, $par2, ...) {  
    instrucciones  
    return valor;  
}
```

Librerías: fichero php que solo contiene funciones entre <?php y ?>, para usar las funciones es necesaria la instrucción include fichero.php; o include_one fichero.php;

Parametros: los **datos primitivos** se pasan por valor, si antepone & delante del parámetro en la definición de los parámetros de la función, se pasa por referencia, **los objetos** siempre se pasan por referencia, los arrays y las cadenas se consideran datos primitivos en el paso de parámetros, así que por defecto siempre se pasan por valor, a no ser que se anteponga & delante, en la definición del parámetro en la cabecera de la función.

```
function nombreFuncion (&$par1, &$par2, ...) {  
    instrucciones  
}
```

Nota: Con las últimas actualizaciones de VSC el anteponer el & delante del parámetro, en la definición de la función, es detectado como error pero funciona correctamente.

Parámetros opcionales:

Para establecer uno o varios parámetros opcionales debemos colocarlos al final, en definición de la función, ya que los valores enviados en la llamada serán recibidos por los parámetros en el orden en que han sido definidos. Si los últimos parámetros definidos no reciben nada en la llamada, serán variables no definidas en la función, a no ser que se establezca un valor por defecto en la definición.

Declaración: function nombreFuncion (\$oblig1, \$oblig2, \$opc1=valor1, \$opc2=valor2, ...) {
Llamada: nombreFuncion (valor1, valor2, ...)

Un truco para establecer un parámetro como opcional en una posición que no sea la última, consistiría en no darle un valor por defecto en la definición de la función, sino que en la llamada se le pasa un valor null y en el código de la función se asigna un valor a ese parámetro que haya recibido valor nulo.

```
function nombreFuncion ($opc1, $opc2, $opc3, ...) {  
    if ($opc1==null) { $opc1=valor1; }  
    ...  
}
```

Sobrecarga:

En PHP no se pueden sobrecargar funciones, podemos hacer un apaño según el número de parámetros recibidos o el tipo de datos, podremos ejecutar distintas acciones:

```
function opera($x, $y, $z) {  
    if (!isset($y)) {  
        return $x * $x;  
    } else if (!isset($z)) {  
        return $x * $y;  
    } else {  
        return $x + $y + $z;}  
}  
  
function opera2($a, $b) {
```

```
if (is_int($a) && is_int($b)) {  
    return $a + $b;  
} else {  
    return $a. ", ". $b;}  
}
```

Diferencia entre include, include_once, require y require_once

– include():

//incluye el fichero functions.php

```
include("functions.php");
```

Esta función nos permite incluir el archivo tantas veces como lo pidamos, sin importar que esté incluido con anterioridad o no, simplemente evalúa el archivo y lo incluye en nuestro documento. En caso de que el archivo no exista, nos devolverá un warning pero el script continuará ejecutándose.

– include_once():

//evalúa si functions.php está incluido, si no lo está lo incluye

```
include_once("functions.php");
```

La diferencia respecto a la función include() y tal y como muestra su nombre, evaluará si el archivo ya ha sido incluido y si es así, no volverá a incluirlo, es decir, se incluye una única vez. En el caso de error por no encontrar el archivo se comporta igual que include().

– require():

//incluye y obliga a que functions.php esté incluido

```
require("functions.php");
```

Posee el mismo comportamiento que include(), la única diferencia reside en el caso de error, pues cuando usamos require(), si el archivo no existe lanza un error fatal que para la ejecución del script.

– require_once():

//evalúa y obliga a estar incluido, si lo está, no hace nada

```
require_once("functions.php");
```

Al igual que ocurre entre include() y include_once(), el comportamiento de require_once() es el mismo que el de require(), tan sólo que el primero evalúa si el archivo ya ha sido incluido y si es así, no vuelve a incluirlo de nuevo. El tratamiento del error es igual al que realiza require().

6.7 Funciones de texto

strcmp (\$cadena1 , \$cadena2) devuelve 0 si son iguales 1 si cadena1 mayor y -1 si es menor en orden alfabético según código ASCII (todas las mayúsculas están antes que todas las minúsculas)

\$saludo = "Hola, estamos trabajando con cadenas"; //la posición del primer carácter es 0

echo "
Todo en minúsculas: " . **strtolower(\$saludo)**;

echo "
Todo en mayúsculas: " . **strtoupper(\$saludo)**;

echo "
Primera letra mayuscula: " . **ucfirst(\$saludo)**;

echo "
Primeras palabras mayúsculas" . **ucwords(\$saludo)**;

echo "
Eliminamos espacios: " . **trim(\$saludo)**;

echo "
Repetimos la cadena: " . **str_repeat(\$saludo, 5)**;

echo "
Contamos los caracteres: " . **strlen(\$saludo)**;

echo "
Busqueda de cadenas: " . **strstr(\$saludo, 'la')**; //devuelve la cadena donde la encuentre hasta el final

echo "
Remplazando cadenas: " . **str_replace(["Hola","Buenas","Hello"], "Adios", \$saludo)**;

//sustituye una o varias cadenas de un array por otra cadena

echo "
Extraer cadenas: " . **substr(\$saludo, 2, 8)**; //extrae desde la posición 2 hasta la 10 (2+8 caracteres), si se omite la cantidad de caracteres extrae hasta el final. Si la posición inicial se indica en negativo se cuenta desde el final de la cadena hacia atrás (-1 es la última posición)

echo "
Devuelve posición primera ocurrencia: ".**mb_strpos(\$saludo, "estamos")**; //si no encuentra devuelve false

echo "
Devuelve posición primera ocurrencia: " . **strpos(\$saludo, "estamos")**; //si no encuentra devuelve cadena vacia

echo "
Devuelve posición última ocurrencia: " . **strrpos(\$saludo, "estamos")**; //si no encuentra devuelve cadena vacia

echo "
Voltea un string dado".**strrev(\$saludo)**; //voltea o invierte una cadena

str_starts_with(\$cadena, \$palabra); // devuelve true o false si la cadena comienza o no con la palabra

str_ends_with(\$cadena, \$palabra); // devuelve true o false si la cadena finaliza o no con la palabra

preg_match("/patron/i", \$saludo); //devuelve 0 si no encuentra o 1 si encuentra el patrón del string en \$saludo sin distinguir mayúsculas, si se quita la i, si distingue mayúsculas y minúsculas. El patrón se define como una expresión regular con caracteres comodín:

"." 1 solo carácter cualquiera excepto "\n"

"?" 0 o 1 ocurrencias del carácter que le precede

"*" cero o más ocurrencias del carácter que le precede

"+" una o más ocurrencias del carácter que le precede

"^" comienza por esa cadena, con "/^patron/m" da nº de líneas coincidentes

"\$" termina por esa cadena con "/\$patron/m" da nº de líneas coincidentes

"[lista de caracteres]" 1 solo carácter de los de la lista (sin separadores)

"[^lista de caracteres]" 1 carácter que no sea ninguno de la lista (sin separadores)

Explicado en url: <https://diego.com.es/expresiones-regulares-en-php>

ord(string): devuelve el código ASCII del primer carácter de la cadena pasada por parámetro

chr(código ascii): devuelve el carácter correspondiente al código ascii pasado por parámetro

\$array = str_split (\$saludo); devuelve un array con cada carácter en una posición

Un **string puede ser tratado como un array**, donde cada carácter de la cadena se encuentra en una posición comenzando en 0.

```
$s="cadena de texto";  
for ($i=0; $i<strlen($s);$i++) { echo $s[$i]. "-"; }
```

nl2br: imprime una cadena con los saltos de línea (si imprimimos un string con saltos de línea necesitamos usar esta función si queremos que se produzca un
 por cada salto de línea en la página web). Ejemplo: `echo nl2br($cadena);`

6.8 Ejercicios de Cadenas

Ejercicio 1.

Imprimir carácter por carácter un string dado, cada uno en una línea distinta.

Ejercicio 2.

Cambiar todas las vocales de la frase “Tengo una hormiguita en la patita, que me esta haciendo cosquillitas y no me puedo aguantar” por otra vocal pedida al usuario.

Ejercicio 3.

Contar cuantas palabras tiene una frase introducida por el usuario, ten en cuenta que el usuario puede poner varios espacios seguidos, incluso al principio o al final.

Ejercicio 4.

Verificar si un string leído por teclado finaliza con la misma palabra que empieza.

Ejercicio 5.

Intercambiar un string dado, hasta volverlo a su forma original:

ejemplo: Hola, ahol, laho, olah, hola (stop).

Ejercicio 6.

Dado un párrafo con dos frases (separadas por un punto), contar cuántas palabras tiene cada frase.

Ejercicio 7.

Verificar si en una frase se encuentra una determinada palabra pedida al usuario.

Ejercicio 8.

Pedir un string al usuario e imprimir todos los números seguidos y sin espacios, correspondientes al código ascii de cada uno de sus caracteres. Posteriormente calcular la frase original a partir de dichos números (usar un array).

Ejercicio 9.

Pedir al usuario una cadena de caracteres e imprimirla invertida.

Ejercicio 10.

Escribir un programa que pida un nombre (con sus apellidos) y escriba en pantalla tanto el nombre con las primeras letras en mayúsculas como las iniciales de dicho nombre.

Ejercicio 11.

Escribir una clase que lea n caracteres que forman un número romano y que imprima:

- a. si la entrada fue correcta, un string que represente el equivalente decimal
- b. si la entrada fue errónea, un mensaje de error.

Nota: La entrada será correcta si contiene solo los caracteres M:1000, D:500, C:100, L:50, X:10, I:1. No se tendrá en cuenta el orden solo se sumará el valor de cada letra.

Ejercicio 12.

Escribir un programa que dado un texto de un telegrama que termina en punto:

- a. contar la cantidad de palabras que posean más de 10 letras
- b. reportar la cantidad de veces que aparece cada vocal
- c. reportar el porcentaje de espacios en blanco.
- d. Nota: Las palabras están separadas por un espacio en blanco.

Ejercicio 13.

Escribir un programa que dado un texto que finaliza en punto, se pide:

- a. la posición inicial de la palabra más larga y su longitud
- b. cuantas palabras con una longitud entre 8 y 16 caracteres poseen más de tres veces la vocal "a"

Nota:

- 1.- Las palabras pueden estar separadas por uno o más espacios en blanco.
- 2.- Puede haber varios espacios en blanco antes de la primera palabra y también después de la última.
- 3.- Se considera que una palabra finaliza cuando se encuentra un espacio en blanco o un signo de puntuación.

6.9 Funciones de fecha

checkdate (\$mes, \$dia, \$año); //devuelve true o false según la fecha sea correcta o no (para el mes y el día admite 1 o 2 dígitos, y para el año 2 o 4 dígitos, sabiendo que con 2 dígitos los valores entre 00-69 hacen referencia a 2000-2069 y 70-99 a 1970-1999).

Funcion date (formato [, fecha]); Devuelve un String con la fecha/hora formateada actual u opcionalmente, una indicada por parámetro en formato fecha (número entero correspondientes a los segundos transcurridos desde 1900).

\$fecha=date("d/m/Y"); //La fecha de hoy es:02/09/2018 (el carácter '/' se puede cambiar)

\$fecha=date("j/n/y"); //La fecha de hoy es:2/9/18 (el carácter '/' se puede cambiar)

\$hora=date("H:i:s"); //La hora actual es:15:06:31 (el carácter ':' se puede cambiar)

Caracteres dentro de la función date () con su aplicación práctica:

a -> Imprime "am" o "pm"

A -> "AM" o "PM"

h -> La hora en formato (01-12)

H -> Hora en formato 24 (00-23)

g -> Hora de 1 a 12 sin un cero delante

G -> Hora de 1 a 23 sin cero delante

i -> Minutos de 00 a 59

s -> Segundos de 00 a 59

d -> Día del mes (01 a 31)

j -> Día del mes sin cero (1 a 31)

w -> Día de la semana (0 a 6). El 0 es el domingo

m -> Mes actual (01 al 12)

n -> Mes actual sin ceros (1 a 12)

Y -> Año con 4 dígitos (2004)

y -> Año con 2 dígitos (04)

z -> Día del año (0 a 365)

t -> Número de días que tiene el mes actual

L -> 1 or 0, según si el año es bisiesto o no

Funcion time(); devuelve un entero que representa la fecha/hora actual del sistema

Funcion strtotime(cadena); convierte un string a fecha/hora (entero que representa un día/hora concreto)

echo strtotime("now"); //Fecha y hora actual

echo strtotime("10 September 2000"); // 10 de septiembre de 2000

echo strtotime("9/15/21"); // 15 de septiembre de 2021 (Fíjate en el orden "m/d/Y")

echo strtotime("2021-9-15"); // 15 de septiembre de 2021 (Fíjate en el orden "y-m-d" o "d-m-y")

echo strtotime("\$mes/\$dia/\$año"); fecha creada a partir de 3 variables (Fíjate en el orden)

echo strtotime("+1 day"); // actual + 1 día

echo strtotime("+1 week"); //actual + 1 semana


```
echo strptime("+1 week 2 days 4 hours 2 seconds"); //actual + 1 semana 2 días 4 hrs y 2 seg
echo strptime("next Thursday"); //próximo jueves
echo strptime("last Monday"); //ultimo lunes que hayamos pasado
```

Sumar días, semanas, meses, años a una fecha

```
$fecha = "22-10-2021";
//En las siguientes, $fecha se podría suprimir y por defecto tomaría la fecha actual
echo date("d-m-Y",strptime("$fecha + 1 days")); //sumo 1 día
echo date("d-m-Y",strptime("$fecha + 1 week")); //sumo 1 semana
echo date("d-m-Y",strptime("$fecha + 1 month")); //sumo 1 mes
echo date("d-m-Y",strptime("$fecha + 1 year")); //sumo 1 año
```

Comparar fechas: Las fechas tienen que ser comparadas en formato fecha UNIX(entero) y si además no quiero tener en cuenta la hora, debo formatearla previamente para establecer la hora a cero.

```
$fecha_actual = strptime(date("d-m-Y 00:00:00",time())); //fecha y hora actual pero con hora a 0
$fecha_entrega = strptime("22-11-2024 00:00:00");
if($fecha_actual > $fecha_entrega){
echo "El plazo de entrega todavía no ha finalizado";
}else{
echo "El plazo de entrega ya ha finalizado";
}
```

6.10 Ejercicios de Fechas

Ejercicio 1.

Crea un formulario donde el usuario introduce una fecha a través de 3 cajas de texto, si no es correcta se debe indicar en un mensaje; si es correcta se debe mostrar en el formato elegido. Crea una lista desplegable con todas las posibilidades de formato que se te ocurran.

Ejercicio 2.

Realiza un ejercicio similar al anterior, pero para la hora.

Ejercicio 3.

Pedir una fecha en un formulario con un input de fecha y mostrar a que día de la semana corresponde (en español).

Ejercicio 4.

Pedir una fecha en un formulario con un input de fecha y mostrarla en el formato "12 de Enero de 2018" (en español).

Ejercicio 5.

Pedir un día de la semana en un formulario, seleccionándolo desde una lista desplegable. Mostrar la fecha correspondiente al próximo día de la semana elegido por el usuario.

Ejercicio 6.

Mostrar el día de la semana que correspondería, una vez transcurridos un número de años, meses, y días elegidos por el usuario, a partir de la fecha actual.

Ejercicio 7.

Pedir al usuario su fecha de nacimiento y una fecha futura, y mostrar la edad que tendrá en esa fecha (un año tiene $60 \cdot 60 \cdot 24 \cdot 365.25$ segundos)

Ejercicio 8.

Pedir la fecha de nacimiento y el nombre de dos personas y mostrar la edad de cada una, así como el nombre de la persona mayor.

7. Sesiones y cookies

7.1 Sesiones

Las sesiones se utilizan en PHP para guardar información en la memoria RAM. Esta información no se pierde si el usuario salta de una página a otra.

Hemos visto anteriormente cómo enviar datos entre dos páginas mediante un formulario, ahora imagina que visitas varias veces la misma página, que saltas a otra, que vuelves de nuevo a la primera; sería muy engorroso estar mandando datos constantemente mediante formularios entre unas páginas y otras. Mediante las sesiones se puede conservar o modificar la información que nos interese independientemente de la/s página/s que se vayan visitando. El valor que se almacena en la memoria está disponible mientras no se cierre la sesión.

Un uso típico de una sesión es el carrito de la compra de una tienda on-line. Podemos visitar todas las páginas que queramos de la tienda e ir añadiendo o quitando productos del carrito gracias a que esta información se graba en una sesión.

La sesión comienza con la función `session_start()` y debe colocarse siempre al principio, antes de mostrar cualquier cosa en el documento HTML.

El ejemplo más sencillo del uso de sesiones es un contador de visitas a una página.

```
<?php session_start(); // Inicio de sesión
    if(isset($_SESSION['visitas'])) {
        $_SESSION['visitas']++;
    } else {
        $_SESSION['visitas'] = 1;
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
    </head>
    <body>
        <?php
            echo "Visitas: ".$_SESSION['visitas'];
        ?>
    </body>
</html>
```

Observa que si `$_SESSION['visitas']` no tiene ningún valor asignado, se crea a la vez que se le asigna el 1, ya que es la primera vez que se visita la página; en caso contrario, se incrementa en 1 su valor de modo que va contando las veces que se carga la página. Incluso si se cierra la pestaña del navegador y se vuelve a abrir otra, el número de visitas continúa donde se quedó.

Para volver a reiniciar el contador de visitas y, por tanto cerrar la sesión, es necesario cerrar el navegador y abrirlo de nuevo.

Dentro del código PHP podemos indicar que queremos cerrar una sesión mediante `session_destroy()`.

En el siguiente ejemplo tenemos dos variables que podemos incrementar, decrementar o poner a cero. El valor de esas variables se guarda en una sesión.

```
<?php session_start(); // Inicia la sesión
// La primera vez que se carga la página, se inicializan
// las variables de sesión a y b a cero.
if(!isset($_SESSION['a'])) {
    $_SESSION['a'] = 0;
}
if(!isset($_SESSION['b'])) {
    $_SESSION['b'] = 0;
}
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
<?php
if (isset($_POST['accion'])) {
    switch($_POST['accion']) {
        case "incA" :
            $_SESSION['a']++; break;
        case "decA" :
            $_SESSION['a']--; break;
        case "incB" :
            $_SESSION['b']++; break;
        case "decB" :
            $_SESSION['b']--; break;
        case "cierra":
            session_destroy();
            header("refresh: 0;"); // refresca la página
    }
}
?>
<h1> a = <?php echo $_SESSION['a'] ?>
<br> b = <?php echo $_SESSION['b'];?> </h1>
<form action="#" method="POST">
<select name="accion">
    <option value="incA">Incrementa a</option>
    <option value="decA">Decrementa a</option>
    <option value="incB">Incrementa b</option>
```

```

        <option value="decB">Decrementa b</option>
        <option value="cierra">Cierra sesión</option>
    </select>
    <input type="submit" value="OK">
</form>
</body>
</html>

```

7.1.1 Cerrar sesión por inactividad

Para aumentar la seguridad en una aplicación web, podemos establecer un tiempo de inactividad, tal que una vez transcurrido dicho periodo sin interacción (recarga de la página), se destruya la sesión y obligue a loguearse de nuevo. Para ello lo único que tenemos que hacer es registrar una variable de sesión `$_session['timeout']` que recoja el instante en que se ha cargado la página gracias a la función `time()`, y al principio en cada recarga solo tenemos que comprobar si los segundos transcurridos desde el último registro en dicha variable de sesión y el instante actual, es superior a los segundos que hayamos establecidos como máximo periodo permitido de inactividad, y en tal caso destruir la sesión y refrescar la página.

```

<?php
    session_start();
    $inactividad = 60;
    if (isset( $_REQUEST['cerrar'])) {
        session_destroy();
        header("refresh: 0;");
    }
    if(isset($_SESSION["timeout"])){ //la primera vez no entra, pues no hay registro previo
        // Calcular el tiempo de vida de la sesión (TTL = Time To Live)
        $sessionTTL = time() - $_SESSION["timeout"];
        if($sessionTTL > $inactividad){
            session_destroy();
            header("refresh: 0;");
        }
    }
?>
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Inactividad</title>
</head>
<body>
    <?php

```

```

        if (isset( $_REQUEST['user'])) {
            $_SESSION['user'] = $_REQUEST['user'];
        }
        if (!isset( $_SESSION['user'])) {
?>
<h3>IDENTÍQUESE</h3>
<form action="" method="post">
    Usuario: <input type="text" name="user">
    <input type="submit" value="Aceptar">
</form>
<?php
    }else {
?>
        <h3>Bienvenido a su página personal, <?= $_SESSION['user'] ?></h3>
        <form action="" method="post">
            <input type="submit" value="Cerrar Sesión" name="cerrar">
        </form>
<?php
    }
    //registramos en la sesión, el momento en que se ha cargado la página
    $_SESSION["timeout"] = time();
?>
</body>
</html>

```

7.2 Cookies

Una cookie (galleta en inglés) es un fichero que se graba en el ordenador del propio usuario, no en el servidor. Permite guardar información de tal forma que no es necesario enviarla mediante formularios al pasar de una página a otra. Una cookie es algo parecido a una sesión aunque, a diferencia de esta última, la cookie se graba en el disco duro del ordenador. Antes de ver como se crean y consultan las cookies en PHP, es conveniente aclarar los siguientes puntos:

- La creación de cookies desde el servidor, en lenguajes como PHP, se debe producir en la respuesta de una solicitud HTTP (Response). O sea, solo cuando devuelvo los datos de una solicitud al servidor soy capaz de escribir una cookie en el navegador.
- La lectura de cookies se realiza a través de la petición. Si el navegador del usuario tiene alguna cookie almacenada en el sistema, llega al servidor en la solicitud emitida por el cliente web.
- Las cookies viajan en las cabeceras del protocolo HTTP, de manera transparente para el usuario. La información para la creación o actualización de cookies se escribe dentro de las cabeceras del HTTP de respuesta. La información para la lectura de las cookies viaja al servidor en las cabeceras HTTP de la solicitud.

- Por todo ello, cuando proceso una página en el servidor que escribe una cookie, esa cookie solo estará presente para lectura en la siguiente solicitud que se realice al servidor.

Las cookies se crean con la función `setcookie()`. El formato de esta función es el siguiente:

```
setcookie(nombre, valor, momento de expiración)
```

El momento de expiración es aquel en el que la cookie se elimina y se expresa en segundos. Normalmente se utiliza la función `time()` junto con el número de segundos que queremos que dure la cookie. Por ejemplo, si queremos crear una cookie de nombre `usuario`, inicializada a `Luis` y que dure una semana, escribiríamos lo siguiente:

```
setcookie("usuario", "Luis", time() + 7*24*60*60)
```

Si no se especifica el momento de expiración, la cookie durará hasta que se cierre el navegador.

Para recuperar el valor de cualquier cookie se utiliza el array `$_COOKIE`. Por ejemplo, para mostrar el valor de la cookie de nombre `usuario`, escribiríamos lo siguiente:

```
echo $_COOKIE["usuario"];
```

A la hora de depurar un programa, es muy útil mostrar todas las cookies. Podemos hacer esto con `print_r($_COOKIE)`.

A continuación se muestra un ejemplo completo. Se trata de una aplicación que muestra nuestra actriz y nuestro actor favorito. Podemos cambiar nuestras preferencias de tal forma que la información permanece en el disco, así se conservan los datos aunque se reinicie el navegador, e incluso aunque se apague y se vuelva a encender el ordenador.

```
<?php
// Si se envían datos desde el formulario de actores,
// se actualizan las cookies
if (isset($_POST["actriz"])) {
    $actriz = $_POST["actriz"];
    $actor = $_POST["actor"];
    setcookie("actriz", $actriz, time() + 3*24*3600);
    setcookie("actor", $actor, time() + 3*24*3600);
} else if (isset($_COOKIE["actriz"])) {
    $actriz = $_COOKIE["actriz"];
    $actor = $_COOKIE["actor"];
}
// Borrado de cookies y variables
if (isset($_POST["borraCookies"])) {
    setcookie("actriz", NULL, -1);
    setcookie("actor", NULL, -1);
    unset($actriz);
    unset($actor);
}
```

```

    }
?>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      if (!isset($actriz)) {
        echo "No has elegido todavía a tus actores favoritos.<br>";
        echo "Utiliza el siguiente formulario para hacerlo.<br>";
      } else {
        echo "<h2>Actriz favorita: ".$actriz."</h2>";
        echo "<h2>Actor favorito: ".$actor."</h2>";
        echo "Introduce nuevos nombres si quieres cambiar tus preferencias.<br>";
      }
    ?>
    <form action="#" method="post">
      Actriz: <input type="text" name="actriz"><br>
      Actor: <input type="text" name="actor"><br>
      <input type="submit" value="Aceptar">
    </form>
    <hr>
    <form action="#" method="post">
      <input type="hidden" name="borraCookies" value="si">
      <input type="submit" value="Borrar cookies"> </form>
    </body>
  </html>

```

Hay que tener en cuenta algo muy importante en cuanto al comportamiento de las cookies que puede dar muchos quebraderos de cabeza: el valor de una cookie se actualiza en la siguiente carga de la página, no tiene el último valor que se le da sino el anterior.

Fíjate cómo se borra una cookie, se utiliza el mismo `setcookie` que se usa para darle un valor, pero esta

vez dándole el valor `NULL`. `setcookie("actriz", NULL, -1);`

Puedes ver las todas las cookies almacenadas en el equipo. Para el navegador Firefox, selecciona Editar → Preferencias → Privacidad → eliminar cookies de forma individual. Puedes buscar las cookies por dominio, así que, filtrando por localhost, podrás ver las que hemos creado con el programa de ejemplo.

7.3 Ejercicios para practicar

Ejercicio 1

Escribe un programa que calcule la media de un conjunto de números positivos introducidos por teclado. A priori, el programa no sabe cuántos números se introducirán. El usuario indicará que ha terminado de introducir los datos cuando meta un número negativo. Utiliza sesiones.

Ejercicio 2

Realiza un programa que vaya pidiendo números hasta que se introduzca un número negativo y nos diga cuantos números se han introducido, la media de los impares y el mayor de los pares. El número negativo sólo se utiliza para indicar el final de la introducción de datos pero no se incluye en el cómputo. Utiliza sesiones.

Ejercicio 3

Escribe un programa que permita ir introduciendo una serie indeterminada de números mientras su suma no supere el valor 10000. Cuando esto último ocurra, se debe mostrar el total acumulado, el contador de los números introducidos y la media. Utiliza sesiones.

Ejercicio 4

Establece un control de acceso mediante nombre de usuario y contraseña para el ejercicio 1 de esta relación. Realiza una nueva versión del ejercicio1, de modo que si lo cargamos sin la sesión iniciada nos redirija a la página de login, y en caso contrario muestre el ejercicio normalmente, también debemos incluir un botón “cerrar sesión” para cerrar la sesión del usuario y volver a la página de login.

Al cargar la página de login, si la sesión está iniciada redirige automáticamente a la página del ejercicio1 y si no, mostrará el formulario de identificación con usuario y contraseña.

Ejercicio 5

Crea una tienda on-line sencilla con un catálogo de productos y un carrito de la compra. Un catálogo de cuatro o cinco productos será suficiente. De cada producto se debe conocer al menos la descripción y el precio. Todos los productos deben tener una imagen que los identifique. Al lado de cada producto del catálogo deberá aparecer un botón Comprar que permita añadirlo al carrito. Si el usuario hace clic en el botón Comprar de un producto que ya estaba en el carrito, se deberá incrementar el número de unidades de dicho producto. Para cada producto que aparece en el carrito, habrá un botón Eliminar por si el usuario se arrepiente y quiere quitar un producto concreto del carrito de la compra. A continuación se muestra una captura de pantalla de una posible solución.

APRENDE PHP CON EJERCICIOS SOLUCIONES A LOS EJERCICIOS

6. Sesiones y Cookies

Tienda on-line *La Estilográfica*

Productos



Pelikan Souvèran M-1000
Precio: 545 €

[Comprar](#)



Parker Duofold International
Precio: 406 €

[Comprar](#)



Visconti Van Gogh
Precio: 180 €

[Comprar](#)

Carrito



Pelikan Souvèran M-1000
Precio: 545 €
Unidades: 2

[Eliminar](#)



Parker Duofold International
Precio: 406 €
Unidades: 1

[Eliminar](#)

Total: 1496 €

Ejercicio 6

Amplía el programa anterior de tal forma que se pueda ver el detalle de un producto. Para ello, cada uno de los productos del catálogo deberá tener un botón Detalle que, al ser accionado, debe llevar al usuario a la vista de detalle que contendrá una descripción exhaustiva del producto en cuestión. Se podrán añadir productos al carrito tanto desde la vista de listado como desde la vista de detalle.

Ejercicio 7

Escribe un programa que guarde en una cookie el color de fondo (propiedad `background-color`) de una página. Esta página debe tener únicamente algo de texto y un formulario para cambiar el color.

Ejercicio 8

Realiza un programa que escoja al azar 5 palabras en inglés de un mini-diccionario. El programa pedirá que el usuario teclee la traducción al español de cada una de las palabras y comprobará si son correctas. Al final, el programa deberá mostrar cuántas respuestas son válidas y cuántas erróneas. La aplicación debe tener una opción para introducir los pares de palabras (inglés - español) que se deben guardar en cookies; de esta forma, si de vez en cuando se dan de alta nuevas palabras, la aplicación puede llegar a contar con un número considerable de entradas en el mini-diccionario.

Ejercicio 9

Amplía el ejercicio 6 de tal forma que los productos que se pueden elegir para comprar, se almacenen en cookies (solo los productos, no almacenar el contenido de la cesta). La aplicación debe ofrecer, por tanto, las opciones de alta, baja y modificación de productos.

Cuando no hay productos almacenados en la cookie, se toman los productos por defecto usados en el ejercicio 6, asignándolos en código y además se almacenan en la cookie, para que estén disponibles para las futuras cargas de la página. Por tanto cada vez que se cargue la página principal, se cargará una array en la sesión con todos los productos de la tienda recuperados de la cookie, y cada vez que se actualicen los productos en dicho array (alta, baja o modificación) también debe actualizarse la cookie para que dichos cambios estén disponibles en la siguiente carga de la página.

También debemos añadir un botón para inicializar los productos por defectos (los del ejercicio 6) deshaciendo todos los cambios realizados en los productos. Añadir un botón Administrar productos en la página principal, que lleve a una segunda página, donde se realizarán todas las operaciones de mantenimiento de los productos.

7.4 Ejercicios para resolver

Ejercicio 1

Crear una página principal con un botón 'Añadir color' para generar un color aleatorio que además se establecerá como color de fondo de la página, cada vez que se pulsa irá generando un color nuevo (actualizando el fondo), que se irán almacenando en un array de sesión. Habrá un segundo botón 'Mostrar paleta creada' que dirige a una segunda página que mostrará una paleta con los colores generados. Esta paleta no es más que una tabla con un máximo de 5 celdas por cada fila, y en cada celda se muestra un color de los generados. Debajo de la tabla tendremos 2 botones uno para volver a la página principal y seguir añadiendo colores a la paleta y otro para destruir la sesión y generar una paleta nueva. Además al pulsar en cada celda de la tabla el color de fondo de la página cambiará al color de la celda pulsada.

Ejercicio 2

Crear una página que simula una encuesta. Se realizará una pregunta, con dos botones para responder, cada vez que se pulse un botón se irán contabilizando (usa sesiones) los votos y actualizando una barra que muestre el número de votos de cada respuesta. Este resultado se irá almacenando también en una cookie, de manera que si se cierra el navegador, al abrir la página de nuevo se mostrarán los resultados hasta el momento en que se cerró. Crear la cookie para 3 meses.

¿Crees que actualmente hay corrupción en el gobierno?

SI 

NO 

Ejercicio 3

Realizar una tienda con un carrito de la compra más completo que el realizado en el boletín. En la página principal tendremos un listado compuesto por una tabla con 4 columnas, nombre del producto, precio, imagen y botón para añadir a la cesta, si se añade más de una vez se aumenta la cantidad del producto en la cesta. También se mostrará cuantos productos hay en la cesta en todo momento y un enlace para acceder a dicha cesta que mostrará otro listado de los productos añadidos y su cantidad, junto a cada producto habrá un botón eliminar que permita quitar una unidad de ese producto y si se llega a 0 se elimina el producto de la cesta. Al final de la cesta se mostrará el importe total de todos los productos y un botón o enlace para cerrar la cesta y volver a la página principal.

Por último, en la página principal al pulsar sobre la imagen de un producto se abrirá en otra página la imagen a tamaño original (algo más grande) junto con los datos del producto y el detalle del mismo (una breve descripción).




Crear manualmente en código, un array de sesión con todos los productos la primera vez que se carga la página en una sesión nueva (con 3 o 4 productos es suficiente). El array puede ser asociativo con clave 'nombre del producto' y valor un array con los valores 'precio, detalle' y la imagen puede coincidir con el nombre del producto más la extensión

Los productos añadidos en la cesta deben almacenarse en una cookie por si se cierra el navegador y se abre de nuevo se recuperen automáticamente los productos añadidos a la cesta.

Página principal

La tiendecita			CESTA 7Prod
Producto	Precio	Imagen	
raton	6		<input type="button" value="Comprar"/>
teclado	11		<input type="button" value="Comprar"/>
monitor	80		<input type="button" value="Comprar"/>
joystick	33		<input type="button" value="Comprar"/>

Página contenido cesta

PRODUCTOS EN TU CESTA			
raton	4	 raton 6 euros	<input type="button" value="Eliminar"/>
teclado	2	 teclado 11 euros	<input type="button" value="Eliminar"/>
monitor	1	 monitor 80 euros	<input type="button" value="Eliminar"/>
Total	7	126	VACIAR CESTA
<input type="button" value="Volver a la tienda"/>			

7.5 Conceptos de la unidad

Sesiones

Al principio, antes de html incluir inicio de sesión:

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

//Para evitar warnings podemos poner

```
<?php if ( session_status() == PHP_SESSION_NONE) { session_start(); }?>
```

Acceso a variables de sesión a través del array asociativo \$_SESSION['variable']

```
if(!isset($_SESSION['visitas'])) { $_SESSION['visitas']=valor;}
```

Destruir sesión (borrar todas las variables de sesión)

```
session_destroy(); //destruye la sesion
```

```
header("refresh: 0;"); // refresca la página, necesario después de destruir la sesión
```

Importante: Trabajando con sesiones se debe utilizar el método POST en formularios.

Cookies

Las cookies se crean y actualizan con la función setcookie() al principio de la página, antes de html, y el valor registrado no será accesible hasta la próxima carga de la página, por lo que además de registrarlo en la cookie deberíamos almacenarlo en una variable para usarlo en la carga actual de la página, además de quedar registrado en la cookie para la próxima carga:

```
setcookie(nombre, valor, segundos hasta expiración);
```

Ejemplo: setcookie("usuario", "Luis", time() + 7*24*60*60);

Acceso a valores almacenados en cookie a través del array asociativo \$_COOKIE['nombre']

```
if (isset($_COOKIE['usuario'])) {$_SESSION['usuario'] = $_COOKIE['usuario'];}
```

Borrar una cookie

```
setcookie("usuario", NULL, -1);
```

8. Ficheros

8.1 Subir un fichero a través de un formulario

A través un 'input' tipo 'file' podemos almacenar un fichero recibido desde un formulario, en una carpeta del servidor. Para ello debemos definir el formulario con un atributo especial que permita la subida de ficheros (enctype="multipart/form-data"). Ejemplo:

```
<form enctype="multipart/form-data" action="destino.php" method="POST">
  <input type="file" name="imagen">
  <input type="submit" value="Añadir">
</form>
```

En la página php que reciba el fichero debemos incluir la siguiente función para almacenar el archivo en la carpeta deseada y con el nombre elegido.

```
move_uploaded_file($_FILES["nombre_en_formulario"]["tmp_name"],
                  "/carpeta_destino/" . $_FILES["nombre_en_formulario"]["name"]);
```

Con la función anterior se mueve el archivo recibido, desde la carpeta donde se ha guardado temporalmente a la nueva ubicación de manera definitiva. El array asociativo \$_FILES recoge de manera automática los ficheros recibidos desde el formulario, por lo que accedemos con su nombre temporal ["tmp_name"] para guardarlo en la carpeta de destino con un nombre nuevo o con su nombre original definido en ["name"]

8.2 Introducción al manejo de archivos con PHP.

Cuando necesitamos crear, guardar, leer o escribir archivos en la creación de sitios web, el manejo de éstos se convierte en una prioridad. Para ello, vamos a ver una serie de funciones y algunos ejemplos para familiarizarnos con el manejo de archivos a nivel básico en php.

FUNCIÓN FOPEN

La función fopen() sirve para abrir ficheros (archivos). Su sintaxis general:

```
<?php //Ejemplo aprenderaprogramar.com
$fp = fopen(fichero, modoDeApertura);
?>
```

Donde \$fp es el descriptor o identificador del fichero abierto que necesitaremos más tarde.

fichero puede ser un archivo de texto, un archivo con extensión .php, o incluso la URL de una página web (por ejemplo http://www.paginaweb.com/). "Abrir" una web con fopen() puede ser muy útil cuando queremos comprobar si la web existe, o si está activa.

Si fichero empieza con "http://", se abre una conexión hacia la web especificada. Siempre debe ponerse una barra (/) al final.

Si fichero empieza con "ftp://", se abre una conexión al servidor especificado.

Si fichero no empieza con ninguna de las cosas anteriormente dichas, se abre una conexión "directa" con el archivo especificado. Si no existe el archivo o dirección especificadas, se devuelve un error.

Hay distintos modos de apertura de archivos, vamos a ver los más habituales.

A la hora de abrir un archivo, hay que saber de qué forma queremos abrirlo. Podemos abrirlo para escritura y lectura, pero de distintas formas:

La función **fopen(fichero, modoDeApertura)** sirve para abrir ficheros (archivos)

Modo	Observaciones
r	Abre el archivo sólo para lectura. La lectura comienza al inicio del archivo.
r+	Abre el archivo para lectura y escritura. La lectura o escritura comienza al inicio del archivo, machacando el contenido previo según se va escribiendo en él.
w	Abre el archivo sólo para escritura. La escritura comienza al inicio del archivo, y elimina el contenido previo. Si el archivo no existe, intenta crearlo.
w+	Abre el archivo para escritura y lectura. La lectura o escritura comienza al inicio del archivo, y elimina el contenido previo. Si el archivo no existe, intenta crearlo.
a	Abre el archivo para sólo escritura. La escritura comenzará al final del archivo, sin afectar al contenido previo. Si el fichero no existe se intenta crear.
a+	Abre el archivo para lectura y escritura. La lectura o escritura comenzará al final del fichero, sin afectar al contenido previo. Si el fichero no existe se intenta crear.

Ejemplos:

```
<?php //Ejemplo aprenderaprogramar.com
$fp = fopen("/apr2/fichero.txt", "r");
$fp = fopen("/apr2/fichero2.txt", "w");
$fp = fopen("http://www.aprenderaprogramar.com/texto.txt", "a+");
$fp = fopen("ftp://ftp.elmundo.es/fichero.txt", "w");
?>
```

Ahora bien, con esta simple instrucción sólo tendremos el archivo abierto para leer, escribir, o leer y escribir. Pero ahora tendremos que hacer uso de las otras instrucciones para escribir o recuperar los datos que estimemos oportunos.

FUNCIÓN FCLOSE

La función **fclose(identificadorDelFichero)** cierra un archivo abierto. Esta función debe ser utilizada después de abrir y manipular el archivo como veremos en los siguientes ejemplos.

LECTURA DE FICHEROS

La función **fgets()** recupera el contenido de una línea de un archivo. Su sintaxis general es:

```
<?php //Ejemplo aprenderaprogramar.com
fgets(descriptorDelFichero);
?>
```

Ejemplo

```
<?php
// Leemos la primera línea de fichero.txt
// fichero.txt tienen que estar en la misma carpeta que el fichero php
// fichero.txt es un archivo de texto normal creado con notepad, por ejemplo.
$fp = fopen("fichero.txt", "r");
$linea = fgets($fp);
fclose($fp);
?>
```

Ahora bien, con esto sólo leeremos la primera línea del fichero de texto, si quisiéramos leer línea a línea hasta el final necesitaríamos el uso de un bucle while, por ejemplo.

```
<?php
// Iremos leyendo línea a línea del fichero.txt hasta llegar al fin (feof($fp))
// fichero.txt tienen que estar en la misma carpeta que el fichero php
// fichero.txt es un archivo de texto normal creado con notepad, por ejemplo.
$fp = fopen("fichero.txt", "r");
while(!feof($fp)) {
    $linea = fgets($fp);
    echo $linea . "<br />";
}
fclose($fp);
?>
```

Esto nos mostrará el contenido del fichero de texto línea a línea. Comprobarás que hemos incluido una nueva función denominada feof (que viene significando algo así como file end of file o “marca de final de archivo”). La sintaxis general para esta función es: feof (identificadorDelArchivo)

La función feof nos devuelve true cuando hemos llegado al final de archivo y false si no lo hemos alcanzado todavía. La condición while (!feof(\$fp)) podríamos haberla escrito también de la siguiente manera: while (feof(\$fp)==false) . Esta sintaxis es equivalente, puedes utilizar la que prefieras, pero debes acostumbrarte a entender ambas formas de escritura.

Finalmente, fíjate como una vez hemos terminado de operar con el fichero escribimos la instrucción fclose(identificadorDelFichero) para cerrar la conexión, cosa que debemos hacer siempre.

ESCRITURA DE FICHEROS

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

```
<?php //Ejemplo aprenderaprogramar.com
```



```
fputs(descriptorDelFichero, cadena);  
?>
```

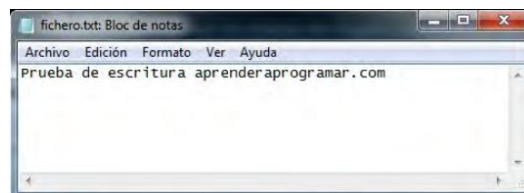
Ejemplo

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

Nota: PHP_EOL (end of line) introduce un salto de línea en PHP. Mediante la concatenación con un punto forzamos el salto de línea después del texto introducido.

Fíjate que al realizar la apertura del fichero en modo w, si el fichero no existe, será creado. Fíjate también como una vez hemos terminado de operar con el fichero escribimos la instrucción fclose(identificadorDelFichero) para cerrar la conexión, cosa que debemos hacer siempre.

Si ahora abrimos el fichero con algún editor de textos como Notepad, bloc de notas o cualquier otro, veremos lo siguiente:



AÑADIR CONTENIDO A UN FICHERO DE TEXTO

A veces no queremos reemplazar el contenido que exista en un fichero de texto, sino añadir un contenido adicional al final de lo que ya exista en el mismo. Vamos a ver cómo podemos hacerlo. A modo de ejemplo añadiremos unas líneas de texto al final del fichero anteriormente escrito, pero esta vez usaremos fwrite(identificadorDelFichero) que escribe una línea en un archivo y es similar a fputs

```
<?php //Ejemplo aprenderaprogramar.com  
$file = fopen("archivo.txt", "a");  
fwrite($file, "Añadimos línea 1" . PHP_EOL);  
fwrite($file, "Añadimos línea 2" . PHP_EOL);  
fclose($file);  
?>
```

Tener en cuenta que el modo de apertura de archivo que hemos usado es a. Si recordamos el significado de este modo:

a: Abre el archivo para sólo escritura. La escritura comenzará al final del archivo sin eliminar el contenido previo existente. Si el fichero no existe se intenta crear.

Ahora podemos ver el fichero creado con un editor de texto cualquiera y observaremos que el contenido no se ha reemplazado, sino que se ha ampliado con una línea nueva.

8.3 Conceptos de la unidad

La función **fgets(identificadorDelFichero[, bytes])** recupera el contenido de una línea, o los bytes (caracteres) indicados opcionalmente o hasta alcanzar el salto de línea.

La función **fgetc(identificadorDelFichero)** recupera un carácter de un archivo.

La función **fgetcsv(\$f, 1000, ",")** es similar a fgets en **archivosCSV**, devolviendo un array con los campos en la línea leída. 1000 es la longitud máxima de la línea o 0 para cualquier longitud, y “,” es el delimitador de campos en el CSV. IMPORTANTE: Si no encuentra el carácter devuelve ‘false’

La función **fputs(identificadorDelFichero, cadena)** escribe una línea en un archivo. (usar constante **PHP_EOL** para salto de línea)

La función **fwrite(identificadorDelFichero, cadena)** escribe una línea en un archivo. (similar a fputs)

La función **fclose(identificadorDelFichero)** cierra un archivo abierto.

La función **ftell(identificadorDelFichero)** devuelve la posición del cursor del archivo (contando todos los caracteres del mismo, incluido los saltos de línea)

La función **rewind(identificadorDelFichero)** posiciona el cursor al principio

La función **file_exists('ruta de fichero')** devuelve booleano si existe fichero o no

La función **unlink('ruta del fichero')** borra el fichero del almacenamiento

La función **is_readable('ruta de fichero')** devuelve booleano si existe fichero y es legible

La función **\$array = file("fichero.txt")** almacena cada línea del fichero en un array

La función **\$cadena=file_get_contents("fichero.txt")** almacena el fichero completo en una cadena eliminando todos los saltos de línea

```
<?php
$fp = fopen("fichero.txt", "r");
while (!feof($fp)) {
    $linea = fgets($fp);
    echo $linea . "<br />";
}
fclose($fp);
?>
<?php
$file = fopen("archivo.txt", "a");
foreach ($array as $cadena) {
    fwrite($file, $cadena . PHP_EOL);
}
fclose($file);
?>
```

8.4 Listar ficheros ubicados en un directorio

```
<?php
$directorio="/Mi unidad/xampp/htdocs/directorio a listar";
$listado=obtenerListadoDeArchivos($directorio);
foreach ($listado as $fichero) {
    echo $fichero['Nombre'];
    echo " <-> ".$fichero['Tamaño'];
    echo " <-> ".date("d/m/Y - H:m a",$fichero['Modificado'])."<br>";
}
function obtenerListadoDeArchivos($directorio){

    // Array en el que obtendremos los resultados
    $res =[];

    // Agregamos la barra invertida al final de la ruta en caso de que no exista
    if(substr($directorio, -1) != "/") $directorio .= "/";

    // Creamos un puntero al directorio y obtenemos el listado de archivos
    $dir = @dir($directorio) or die("Error accediendo al directorio $directorio");
    while(($archivo = $dir->read()) !== false) {
        // Obviamos los archivos ocultos y directorios
        if($archivo[0] == "." || is_dir($directorio . $archivo)) continue;
        if (is_readable($directorio . $archivo)) { //si el archivo es legible
            $res[] = [
                "Nombre" => $archivo,
                "Tamaño" => filesize($directorio . $archivo),
                "Modificado" => filemtime($directorio . $archivo)
            ];
        }
    }
    $dir->close();
    return $res;
}
```

8.5 Ejercicios para practicar

Crea las siguientes funciones en PHP:

Ejercicio 1.

Una función (tipo procedimiento, no hay valor devuelto) denominada escribirTresNumeros que reciba tres números enteros como parámetros y proceda a escribir dichos números en tres líneas en un archivo denominado datosEjercicio.txt. Si el archivo no existe, debe crearlo.

Ejercicio 2.

Una función denominada obtenerSuma (tipo función, devolverá un valor numérico) que reciba una ruta de archivo como parámetro, lea los números existentes en cada línea del archivo, y devuelva la suma de todos esos números.

Ejercicio 3.

Una función denominada obtenerArrNum (tipo función, devolverá un array de valores numéricos) que reciba una ruta de archivo como parámetro, lea los números existentes en cada línea del archivo, y devuelva un array cuyo

índice 0 contendrá el número existente en la primera línea, cuyo índice 1 contendrá el número existente en la segunda línea y así sucesivamente (no usar la función file).

Ejercicio 4.

Crea código php donde a través de la función `escribirTresNumeros` escribas en el fichero los números 2, 8, 14. Luego, mediante la función `obtenerSuma` muestra por pantalla el resultado de sumar los números existentes en el archivo. Finalmente, mediante la función `obtenerArrNum` obtén el array, recórrelo y muestra cada uno de los elementos del array.

Ejercicio 5.

Una función (tipo procedimiento, no hay valor devuelto) denominada `escribirNumerosMod` que reciba dos parámetros: un array de valores enteros y una cadena de texto que puede ser "sobreescribir" ó "ampliar". La función debe proceder a: escribir cada uno de los números que forman el contenido del array en una línea de un archivo `datosEjercicio.txt` usando el modo de operación que se indique con el otro parámetro. Si el archivo no existe, debe crearlo.

Ejemplo: El array que se pasa es `$numeros = array(5, 9, 3, 22);` y la invocación que se utiliza es `escribirNumerosMod($numeros, "sobreescribir");` En este caso, se debe eliminar el contenido que existiera previamente en el archivo y escribir en él 4 líneas, cada una de las cuales contendrá los números 5, 9, 3 y 22.

Ejercicio 6.

Una función (tipo procedimiento, no hay valor devuelto) denominada `leerContenidoFichero` que reciba como parámetro la ruta del fichero y muestre por pantalla el contenido de cada una de las líneas del fichero.

Ejercicio 7.

Crea código php donde a través de la función `escribirNumerosMod` escribas en el fichero los números 2, 8, 14. Luego, mediante la función `leerContenidoFichero` muestra el contenido del fichero. Ahora con la función `escribirNumerosMod` amplía el contenido del fichero y añádele los números 33, 11 y 16. Muestra nuevamente el contenido del fichero por pantalla. Finalmente, escribe el fichero pasándole un array con los números 4, 99, 12 y parámetro <<sobreescribir>> para eliminar los datos que existieran previamente. Muestra el contenido del fichero por pantalla y un mensaje de despedida.

Crea las siguientes páginas en PHP

Ejercicio 8.

Crear una página que permita generar un archivo de texto con las líneas que se vayan escribiendo a través de un formulario de manera indefinida, hasta que se pulse un botón terminar, y a continuación mostrar el contenido del fichero completo en la página.

Ejercicio 9.

Escribe un programa que guarde en un fichero el contenido de otros dos ficheros, de tal forma que en el fichero resultante aparezcan las líneas de los primeros dos ficheros mezcladas, es decir, la primera línea será del primer fichero, la segunda será del segundo fichero, la tercera será la siguiente del primer fichero, etc. Los nombres de los dos ficheros origen y el nombre del fichero destino se deben pasar a través de un formulario. Hay que tener en cuenta que los ficheros de donde se van cogiendo las líneas pueden tener tamaños diferentes.

Ejercicio 10.

Realiza un programa que sea capaz de ordenar alfabéticamente las palabras contenidas en un fichero de texto. El nombre del fichero que contiene las palabras se debe pasar a través de un formulario. El nombre del fichero resultado debe ser el mismo que el original añadiendo la coletilla sort, por ejemplo palabras_sort.txt. Suponemos que cada palabra ocupa una línea.

Ejercicio 11.

Realiza un programa que diga cuántas ocurrencias de un texto hay en un fichero. Tanto el nombre del fichero como el texto se deben pasar como argumentos en la línea de comandos.

Ejercicio 12.

Escribe un programa capaz de quitar las etiquetas html (<etiqueta>) de un documento web. Por ejemplo como en esta línea: <h1>Título</h1> -> Título

Crea un fichero con mismo nombre y la coletilla "Sin" al final, que contiene el contenido del documento original pero sin etiquetas html.

8.6 Ejercicios para resolver

Ejercicio 1.

Crear una aplicación web para mantener un fichero mascotas.txt de los animales inscritos diariamente en una clínica veterinaria, con la siguiente estructura:

```
#02-03-2019#  
pepe-canario-2  
luna-perro-4  
duque-gato-6  
#15-11-2019#  
princesa-hamster-1  
venus-perro-12  
...
```

Al entrar en la aplicación, la fecha seleccionada automáticamente es la fecha actual, por lo que las mascotas son grabadas forzosamente en el día en que nos encontramos.

Crear una página para añadir mascotas en un array de sesión donde la clave es el nombre de la mascota y el valor es otro array con los datos correspondientes al tipo de animal y su edad. Las mascotas que se van añadiendo se van mostrando en una tabla.

Incluir un formulario para añadir los datos de una nueva mascota.

Incluir un botón para grabar todas las mascotas añadidas en el fichero mascotas.txt, con la cabecera de la fecha actual tal como se ve en el ejemplo (el fichero debe mantener la información previa y añadir las líneas de las mascotas nuevas al final del mismo), limpiando la tabla de mascotas añadidas hasta ese momento. Si no se han añadido mascotas previamente ese mismo día habrá que incluir la cabecera con la fecha actual, pero en caso contrario solo hay que añadir las mascotas al final, sin duplicar la cabecera.

Ejercicio 2.

Diseñar una página que muestre un cuadro 'select' con todas las fechas disponibles en el fichero del ejercicio anterior, de manera que al seleccionar y enviar una fecha, se cargue en un array de sesión las mascotas almacenadas en la fecha seleccionada y las muestre en una tabla.

La opción de elegir una fecha siempre estará disponible para poder mostrar las mascotas de la fecha que interese, es decir que se puede ir cambiando de fecha y se actualizan los datos de la tabla.

Nota: al leer una línea de un fichero se añaden espacios al principio o al final, así que para hacer comparaciones debes asegurarte de quitar esos espacios.

Ejercicio 3.

Modifica el ejercicio del carrito de la compra del tema anterior, para que los productos se almacenen en un fichero. Debes crear una página para administrar (insertar y eliminar) los productos de la tienda (que están almacenados en dicho fichero). Puedes trabajar con los productos en un array de sesión, pero cuando se añada o se borre un producto de la tienda, será necesario actualizar el fichero. También se debe seguir guardando la cesta de la compra en una cookie, de manera que se pueda retomar la compra, aunque se cierre el navegador, aunque esa funcionalidad ya la tenemos implementada.

9. Programación orientado a objetos

9.1 El paradigma de la Programación Orientada a Objetos

Mientras que la programación estructurada se basa en la utilización de estructuras de control como las sentencias `if` y los bucles `for` y `while`, y usa como almacenamiento de información las variables y los `arrays`, la programación orientada a objetos se basa, como su nombre indica, en la utilización de objetos.

La programación estructurada y la POO no son excluyentes, un programa basado en objetos seguirá teniendo variables, bucles y sentencias condicionales, no obstante, éste último estará seguramente mejor organizado y será más legible y escalable.

Un objeto en términos de POO no se diferencia mucho de lo que conocemos como un objeto en la vida real. Pensemos por ejemplo en un coche. Nuestro coche sería un objeto concreto de la vida real, igual que el coche del vecino, o el coche de un compañero de trabajo, o un deportivo que vimos por la calle el fin de semana pasado... Todos esos coches serían objetos concretos que podemos ver y tocar. Usando la terminología de la POO diríamos que son instancias.

Tanto mi coche como el coche del vecino tienen algo en común, ambos son coches. En este caso mi coche y el coche del vecino serían instancias (objetos) y coche (a secas) sería una clase. La palabra coche define algo genérico, es una abstracción, no es un coche concreto, sino que hace referencia a algo que tiene una serie de propiedades como matrícula, marca, modelo, color, etc. Este conjunto de propiedades se denominan atributos o variables de instancia.

Clase

Concepto abstracto que denota una serie de cualidades, por ejemplo coche.

Instancia

Objeto palpable, que se deriva de la concreción de una clase, por ejemplo mi coche.

Atributos

Conjunto de características que comparten los objetos de una clase, por ejemplo para la clase coche tendríamos matrícula, marca, modelo, color y número de plazas.

9.2 Encapsulamiento y ocultación

Uno de los pilares en los que se basa la Programación Orientada a Objetos es el encapsulamiento. Básicamente, el encapsulamiento consiste en definir todas las propiedades y el comportamiento de una clase dentro de esa clase; es decir, en la clase `Coche` estará definido todo lo concerniente a la clase `Coche` y en la clase `Libro` estará definido todo lo que tenga que ver con la clase `Libro`.

El encapsulamiento parece algo obvio, casi de perogrullo, pero hay que tenerlo siempre muy presente al programar utilizando clases y objetos. En alguna ocasión puede que estemos tentados a mezclar parte de una clase con otra clase distinta para resolver un problema puntual. No hay que caer en esa trampa. Se

deben escribir los programas de forma que cada cosa esté en su sitio. Sobre todo al principio, cuando definimos nuestras primeras clases, debemos estar pendientes de que todo está definido donde corresponde.

La ocultación es una técnica que incorporan algunos lenguajes que permite esconder los elementos que definen una clase, de tal forma que desde otra clase distinta no se pueden “ver las tripas” de la primera. La ocultación facilita, como veremos más adelante, el encapsulamiento.

9.3 Implementación de clases en PHP

Las clases en PHP comienzan con una letra mayúscula. Es muy recomendable separar la implementación de las clases del programa principal en ficheros diferentes. Desde el programa principal se puede cargar la clase mediante `include` o `include_once` seguido del nombre del fichero de clase. El nombre de la clase debe coincidir con el nombre del fichero que la implementa (con la extensión `.php`).

A continuación, tenemos un ejemplo muy sencillo. Se trata de la implementación de la clase `Persona`. Esta clase tendrá dos atributos: `nombre` y `profesión`.

Se define también el constructor. Este método es muy importante ya que se llamará siempre que se creen nuevos objetos de la clase y servirá generalmente para inicializar los valores de los atributos. En PHP, el constructor de una clase se define con el nombre `__construct()`

Se crea además un método que, aplicado a una instancia de la clase `Persona`, muestra un mensaje por pantalla.

```
<?php class Persona {
    private $nombre;
    private $profesion;
    // Constructor
    public function __construct($nom, $pro) {
        $this->nombre = $nom;
        $this->profesion = $pro;
    }
    public function presentarse() {
        echo "Hola, me llamo " . $this->nombre . " y soy " . $this->profesion . "<br>";
    }
}
```

Observa que los atributos se declaran privados y los métodos se declaran públicos. Eso quiere decir que los atributos serán accesibles únicamente desde la implementación de la clase y que los métodos se podrán utilizar en cualquier lugar siempre que se apliquen a los objetos de la clase adecuada, en este caso a las instancias de `Persona`. Salvo que se indique lo contrario, seguiremos esta regla al realizar los ejercicios.

En el programa principal `index.php` se carga el fichero con la implementación de la clase, se crean dos instancias de la clase `Persona` y se les aplica el método `presentarse`. Mediante la función `var_dump()` se puede visualizar el tipo y el valor de todos los atributos que tiene un objeto concreto.


```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'Persona.php';
      $unTipo = new Persona("Pepe Pérez", "albañil");
      $unNota = new Persona("Rigoberto Peláez", "programador");
      $unTipo->presentarse();
      $unNota->presentarse();
      var_dump($unNota);
      var_dump($unTipo);
    ?>
  </body>
</html>

```

Vamos a mejorar un poco nuestra clase `Persona`. Es poco elegante y sobre todo poco práctico hacer un `echo` desde un método. Es mejor devolver una cadena de caracteres para después procesarla en el programa principal aplicándole estilos, guardándola en una variable, etc.

```

<?php class Persona {
  private $nombre;
  private $profesion;
  // Constructor
  public function __construct($nom, $pro) {
    $this->nombre = $nom;
    $this->profesion = $pro;
  }
  public function presentarse() {
    return "Hola, me llamo " . $this->nombre . " y soy " . $this->profesion . "<br>";
  }
}

```



Recuerda que los ficheros que contienen únicamente código PHP comienzan con la etiqueta `<?php` pero no es necesario que terminen con `?>`. Es más, se recomienda no usar la etiqueta de cierre.

Ahora, el programa que utiliza la clase `Persona` sería ligeramente diferente.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      include_once 'Persona.php';

```

```

$unTipo = new Persona("Pepe Pérez", "albañil");
$unNota = new Persona("Rigoberto Peláez", "programador");
echo $unTipo->presentarse();
echo $unNota->presentarse();
?>
</body>
</html>

```

El constructor es un método especial y por ello comienza con doble carácter de subrayado. Hay otro método especial muy útil, se trata de `__toString`. Si se define el método `__toString` en una clase, cuando se realice un `echo` sobre un elemento de esa clase, se ejecutará dicho `__toString`.

En la siguiente versión de la clase `Persona` se muestra la implementación del método `__toString`.

```

<?php class Persona {
    private $nombre;
    private $profesion;
    private $edad;
    // Constructor
    public function __construct($nom, $pro, $edad) {
        $this->nombre = $nom;
        $this->profesion = $pro;
        $this->edad = $edad;
    }
    public function presentarse() {
        return "Hola, me llamo " . $this->nombre . " y soy " . $this->profesion . "<br>";
    }
    public function __toString() {
        return "<hr><b>$this->nombre</b><br>
            Profesión: $this->profesion<br>
            Edad: $this->edad<hr>";
    }
}

```

En el programa principal `-index.php-` se ejecuta el método `__toString` haciendo `echo` sobre el objeto de la clase `Persona`.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
            include_once 'Persona.php';
            $unTipo = new Persona("Pepe Pérez", "albañil", 30);
            $unNota = new Persona("Rigoberto Peláez", "programador");

```

```

        echo $unTipo;
        echo $unNota;
    ?>
</body>
</html>

```

Cuando se implementa una clase en PHP, igual que se hace con otros lenguajes - es habitual crear métodos getter y setter. Se trata de métodos muy simples. El cometido de un getter es proporcionar (devolver) el valor de un atributo mientras que la misión de un setter es darle un valor a un atributo; este valor que se quiere asignar se pasa como parámetro.



Un elemento `public` (público) es visible desde cualquier clase, un elemento `protected` (protegido) es visible desde la clase actual y desde todas sus subclases (en el siguiente apartado se estudia el concepto de subclase) y, finalmente, un elemento `private` (privado) únicamente es visible dentro de la clase actual. Por regla general, se suelen definir los atributos como privados y los métodos como públicos.

A continuación se muestra la implementación de la clase `Gato`. El método `getSexo()` es un getter que devuelve el valor del atributo `$sexo` (macho o hembra).

```

<?php class Gato {
// atributos
private $color;
private $raza;
private $edad;
private $peso;
private $sexo;
// métodos
public function __construct($s) {
    $this->sexo = $s;
}
public function getSexo() {
    return $this->sexo;
}
public function maulla() {
    echo "Miauuuu<br>";
}
public function ronronea() {
    echo "mrrrrrr<br>";
}
public function come($comida) {
    if ($comida == "pescado") {
        echo "Hmmm, gracias<br>";
    } else {
        echo "Lo siento, yo solo como pescado<br>";
    }
}
public function peleaCon($contrincante) {

```

```

        if (($this->getSexo()) == "hembra") {
            echo "no me gusta pelear<br>";
        } else if (($contrincante->getSexo()) == "hembra") {
            echo "no peleo contra gatitas<br>";
        } else {
            echo "ven aquí que te vas a enterar<br>";
        }
    }
}

```

El programa que prueba la clase `Gato` es el siguiente.

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<?php
    include_once 'Gato.php';
    $garfield = new Gato("macho");
    echo "hola gatito<br>";
    $garfield->maulla();
    echo "toma tarta<br>";
    $garfield->come("tarta selva negra");
    echo "toma pescado, a ver si esto te gusta<br>";
    $garfield->come("pescado");
    $tom = new Gato("macho");
    echo "Tom, toma sopita de verduras<br>";
    $tom->come("sopa de verduras");
    $lisa = new Gato("hembra");
    echo "gatitos, a ver cómo maulláis<br>";
    $garfield->maulla();
    $tom->maulla();
    $lisa->maulla();
    $garfield->peleaCon($lisa);
    $lisa->peleaCon($tom);
    $tom->peleaCon($garfield);
?>
</body>
</html>

```

9.4 Herencia

La herencia es una de las características más importantes de la POO. Si definimos una serie de atributos y métodos para una clase, al crear una subclase, todos estos atributos y métodos siguen siendo válidos.

A continuación se muestra la implementación de la clase `Animal`. Uno de los métodos de esta clase es `duerme`. Luego crearemos las clases `Gato` y `Ave` como subclases de `Animal`. De forma automática, se podrá utilizar el método `duerme` con las instancias de las clases `Gato` y `Ave` ¿no es fantástico?

```
<?php abstract class Animal {  
  
    private $sexo;  
  
    public function __construct($s = "macho") {  
        $this->sexo = $s;  
    }  
    public function __toString() {  
        return "Sexo: $this->sexo";  
    }  
    public function getSexo() {  
        return $this->sexo;  
    }  
    public function duerme() {  
        return "Zzzzzzz";  
    }  
    public function aseate() {  
        return "Me gusta asearme, soy un animal.<br>";  
    }  
}
```

Observa que la definición de la clase `Animal` comienza con la siguiente línea. `abstract class Animal{`
Por tanto, `Animal` será una clase abstracta.



Clase abstracta (**abstract**)

Una clase abstracta es aquella que no va a tener instancias de forma directa, aunque sí habrá instancias de las subclases (siempre que esas subclases no sean también abstractas). Por ejemplo, si se define la clase `Animal` como abstracta, no se podrán crear objetos de la clase `Animal`, es decir, no se podrá hacer `$mascota = new Animal()`, pero sí se podrán crear instancias de la clase `Gato`, `Ave` o `Pinguino` que son subclases de `Animal`.

La clase `Ave` es subclase de `Animal` y la clase `Pinguino`, a su vez, sería subclase de `Ave` y por tanto hereda todos sus atributos y métodos.

Para crear en PHP una subclase de otra clase existente se utiliza la palabra reservada `extends`. A continuación se muestra el código de las clases `Gato`, `Ave` y `Pinguino`, así como el programa que prueba estas clases creando instancias y aplicándoles métodos.

```
<?php  
  
include_once 'Animal.php';
```

```

class Gato extends Animal {
    private $raza;
    public function __construct($s, $r) {

        parent::__construct($s);
        if (isset($r)) {
            $this->raza = $r;
        } else {
            $this->raza = "siamés";
        }
    }
    public function __toString() {
        return parent::__toString() . "<br>Raza: $this->raza";
    }
    public function maulla() {
        return "Miauuuu<br>";
    }
    public function ronronea() {
        return "mrrrrrr<br>";
    }
    public function come($comida) {
        if ($comida == "pescado") {
            return "Hmmm, gracias<br>";
        } else {
            return "Lo siento, yo solo como pescado<br>";
        }
    }
    public function peleaCon($contrincante) {
        if (($this->getSexo()) == "hembra") {
            echo "no me gusta pelear<br>";
        } else if (($contrincante->getSexo()) == "hembra") {
            echo "no peleo contra gatitas<br>";
        } else {
            echo "ven aquí que te vas a enterar<br>";
        }
    }
}

```

Observa cómo se indica que la clase `Gato` es subclase de `Animal`. `class Gato extends Animal`

Para que el programa reconozca que existe la clase `Animal` es necesario incluirla en el archivo actual.

```
include_once 'Animal.php';
```

Mediante `parent` se puede hacer una llamada al método de la clase padre. Por ejemplo `parent::__construct($s);` dentro de la definición de la clase `Gato` invoca al constructor de la clase padre, es decir, la clase `Animal`.

A continuación tenemos la definición de la clase `Ave`, que es una subclase de `Animal`.

```
<?php
include_once 'Animal.php';
class Ave extends Animal {
    public function __construct($s) {
        parent::__construct($s);
    }
    public function aseate() {
        return "Me estoy limpiando las plumas<br>" . parent::aseate();
    }
    public function vuela() {
        return "Estoy volando<br>";
    }
}
```

El siguiente código corresponde a la definición de la clase `Pinguino` que es subclase de `Ave` y, por lo tanto, también es subclase de `Animal`. Observa cómo se sobrescribe el método `vuela()`. Cuando se define un método en una subclase con el mismo nombre que tiene en la superclase, tiene preferencia en la ejecución el método de la subclase.

```
<?php
include_once 'Ave.php';
class Pinguino extends Ave {
    public function __construct($s) {
        parent::__construct($s);
    }
    public function aseate() {
        return parent::aseate() . "A los pingüinos nos gusta asearnos<br>";
    }
    public function vuela() {
        return "No puedo volar<br>";
    }
}
```

Observa que no es necesario incluir el fichero `Animal.php` en la definición de la clase `Pinguino` ya que está incluido en la definición de la clase `Ave`.

Por último mostramos el programa que prueba las clases definidas anteriormente.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <?php
        include_once 'Animal.php'; // no es necesario incluirla
```

```

include_once 'Ave.php'; // no es necesario incluirla
include_once 'Pinguino.php';
include_once 'Gato.php';
$garfield = new Gato("macho", "romano");
$tom = new Gato("macho");
$lisa = new Gato("hembra");
$silvestre = new Gato();
echo "$garfield<hr>";
echo "$tom<hr>";
echo "$lisa<hr>";
echo "$silvestre<hr>";
$miLoro = new Ave();
echo $miLoro->aseate();
echo $miLoro->vuela();
$pingu = new Pinguino("hembra");
echo $pingu->aseate();
echo $pingu->vuela();
?>
</body>
</html>

```

No es necesario incluir las siguientes líneas:

```

include_once 'Animal.php';
include_once 'Ave.php';

```

Ya que los ficheros `Animal.php` y `Ave.php` se cargan desde `Gato.php` y `Pinguino.php` respectivamente. No obstante, se pueden dejar por claridad, para indicar todas las clases que intervienen en el programa.

9.5 Atributos y métodos de clase (`static`)

Hasta el momento hemos definido atributos de instancia como `$raza` o `$sexo` y métodos de instancia como `maulla()`, `come()` o `vuela()`. De tal modo que, si en el programa se crean 20 gatos, cada uno de ellos tiene su propia raza y puede haber potencialmente 20 razas diferentes. También podría aplicar el método `maulla()` a todos y cada uno de esos 20 gatos.

No obstante, en determinadas ocasiones, nos puede interesar tener atributos de clase (variables de clase) y métodos de clase. Cuando se define una variable de clase solo existe una copia del atributo para toda la clase y no una para cada objeto. Esto es útil cuando se quiere llevar la cuenta global de algún parámetro. Los métodos de clase se aplican a la clase y no a instancias concretas.

A continuación se muestra un ejemplo que contiene la variable de clase `$kilometrajeTotal`. Si bien cada coche tiene un atributo `$kilometraje` donde se van acumulando los kilómetros que va recorriendo, en la variable de clase `$kilometrajeTotal` se lleva la cuenta de los kilómetros que han recorrido todos los coches que se han creado.

También se crea un método de clase llamado `getKilometrajeTotal()` que simplemente es un getter para la variable de clase `$kilometrajeTotal`.

```
<?php class Coche {  
  
    // atributo de clase  
    private static $kilometrajeTotal = 0;  
    // método de clase  
    public static function getKilometrajeTotal() {  
        return Coche::$kilometrajeTotal;  
    }  
    private $marca;  
    private $modelo;  
    private $kilometraje;  
    public function __construct($ma, $mo) {  
        $this->marca = $ma;  
        $this->modelo = $mo;  
        $this->kilometraje = 0;  
    }  
    public function getKilometraje() {  
        return $this->kilometraje;  
    }  
    public function recorre($km) {  
        $this->kilometraje += $km;  
        Coche::$kilometrajeTotal += $km;  
    }  
}
```

El atributo `$kilometrajeTotal` almacena el número total de kilómetros que recorren todos los objetos de la clase `Coche`, es un único valor, por eso se declara como `static`. Por el contrario, el atributo `$kilometraje` almacena los kilómetros recorridos por un objeto concreto y tendrá un valor distinto para cada uno de ellos. Si en el programa principal se crean 20 objetos de la clase `Coche`, cada uno tendrá su propio `$kilometraje`.

A continuación se muestra el programa que prueba la clase `Coche`.

```
<!DOCTYPE html>  
<html>  
    <head>  
        <meta charset="UTF-8">  
        <title></title>  
    </head>  
    <body>  
        <?php  
            include_once 'Coche.php';  
            $cocheDeLuis = new Coche("Saab", "93");  
            $cocheDeJuanK = new Coche("Toyota", "Avensis");  
            $cocheDeLuis->recorre(30);
```

```

$cocheDeLuis->recorre(40);
$cocheDeLuis->recorre(220);
$cocheDeJuanK->recorre(60);
$cocheDeJuanK->recorre(150);
$cocheDeJuanK->recorre(90);
echo "El coche de Luis ha recorrido ".$cocheDeLuis->getKilometraje()."Km<br>"; echo
"El coche de Juan Carlos ha recorrido ".$cocheDeJuanK->getKilometraje()."Km<br>";
echo "El kilometraje total ha sido de ".Coche::getKilometrajeTotal()."Km";
?>
</body>
</html>

```

El método `getKilometrajeTotal()` se aplica a la clase `Coche` por tratarse de un método de clase (método `static`). Este método no se podría aplicar a una instancia, de la misma manera que un método que no sea `static` no se puede aplicar a la clase sino a los objetos.

También debemos tener en cuenta que la carga de una página implica que todas las variables definidas en la misma son inicializadas, perdiendo el valor que tuvieran de una carga anterior, por lo que para retener el valor de los atributos estáticos de la clase, tendremos que usar sesiones. Por tanto si la página que hace uso de la clase no realiza ninguna recarga, podremos usar los atributos estáticos explicados anteriormente, pero en caso contrario, debemos definir los atributos estáticos como variables de sesión dentro de la clase, consiguiendo así que no se pierdan sus valores con una recarga. Si definimos los atributos estáticos como variables de sesión en la clase anterior, quedaría de la siguiente forma:

```

<?php
if (session_status() == PHP_SESSION_NONE) session_start();
// atributo de clase
if (!isset($_SESSION['kilometrajeTotal'])) {
    $_SESSION['kilometrajeTotal']=0;
}
class Coche {
    public static function getKilometrajeTotal() {
        return $_SESSION['kilometrajeTotal'];
    }
    private $marca;
    private $modelo;
    private $kilometraje;
    public function __construct($ma, $mo) {
        $this->marca = $ma;
        $this->modelo = $mo;
        $this->kilometraje = 0;
    }
    public function getKilometraje() {
        return $this->kilometraje;
    }
    public function recorre($km) {
        $this->kilometraje += $km;
        $_SESSION['kilometrajeTotal'] += $km;
    }
}

```

9.6 Almacenar objetos en sesiones

Consideremos la siguiente clase:

```
<?php
class MonstruoDeLasGalletas {
    private $galletas; // galletas comidas
    public function __construct() {
        $this->galletas = 0;
    }
    public function getGalletas() {
        return $this->galletas;
    }
    public function come($g) {
        $this->galletas = $this->galletas + $g;
    }
}
```

Para evitar errores al almacenar objetos en variables de sesión hay que tener en cuenta **3 aspectos MUY IMPORTANTES**:

1. En la página que hace uso de la clase y en la que vamos a ir almacenando los objetos creados en sesiones, **la sentencia include de la clase debe ir antes que la sentencia session_start()**, ya que en caso contrario no se reconocerían los objetos almacenados de la clase al recuperarlos en la sentencia 'session_start()'. Ejemplo:

```
<?php
include_once 'MonstruoDeLasGalletas.php';
session_start();
if (!isset($_SESSION['coco'])) {
    $_SESSION['coco'] = new MonstruoDeLasGalletas();
}
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<?php
    if (isset($_REQUEST['numeroDeGalletas'])) {
        $_SESSION['coco']->come($_REQUEST['numeroDeGalletas']);
    }
?>
    <h2>Soy Coco y he comido <?= $_SESSION['coco']->getGalletas(); ?> galletas.</h2>
    <form action="" method="POST">
        Nº de galletas:
        <input type="number" name="numeroDeGalletas" min="1">
        <input type="submit" value="Comer">
    </form>
</body>
</html>
```

2. Cuando almacenemos objetos en variables de sesión, **evitar definir el método mágico `__toString()` en la clase**, ya que en ese caso, incluso al hacer el include de la clase antes de la sentencia `session_start()`, nos saltaría un error de objeto incompleto al recuperar la sesión.

3. Ante cualquier problema al recuperar los objetos almacenados en un array de sesión, podemos evitarlos almacenando los objetos serializados, y para ello debemos serializar el objeto antes de almacenarlo en la sesión y des-serializarlo al recuperarlo. En el proceso de serialización, el objeto queda transformado en una cadena de caracteres, por lo que para recuperar el objeto a partir de la sesión, habrá que hacer el proceso inverso, es decir des-serializar la variable de sesión para obtener el objeto original.

Ejemplo:

```
<?php
include_once 'MonstruoDeLasGalletas.php';
session_start();
if (!isset($_SESSION['coco'])) {
    // Creamos el objeto y lo almacenamos serializado
    $_SESSION['coco'] = serialize(new MonstruoDeLasGalletas());
}
?>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title></title>
</head>
<body>
<?php
if (isset($_POST['numeroDeGalletas'])) {
    $numeroDeGalletas = $_POST['numeroDeGalletas'];
    $coco = unserialize($_SESSION['coco']); // Obtenemos el objeto unserializando
    $coco->come($numeroDeGalletas); // Hacemos las operaciones necesarias
    $_SESSION['coco'] = serialize($coco); // Volvemos a almacenar el objeto serializado
}
?>
<h2>Soy Coco y he comido <?=$coco->getGalletas(); ?> galletas.</h2>
<form action="index.php" method="POST"> Nº de galletas:
    <input type="number" name="numeroDeGalletas" min="1">
    <input type="submit" value="Comer">
</form>
</body>
</html>
```

En la primera carga de página se crea un objeto de la clase `MonstruoDeLasGalletas`, se serializa ese objeto y se guarda en una variable de sesión con nombre `$_SESSION['coco']`.

Antes de operar con el objeto, debe extraerse de la sesión y a partir de ese momento, `$coco` es un objeto de la clase `MonstruoDeLasGalletas` al que le podemos aplicar cualquiera de los métodos definidos. Y por último con el fin de conservar el objeto en la sesión, lo serializamos y lo guardamos de nuevo en el array `$_SESSION`.

9.7 Ejercicios para resolver

Ejercicio 1

Confeccionar una clase Empleado con los atributos nombre y sueldo.

Definir un método “asigna” que reciba como dato el nombre y sueldo y actualice los atributos (debe comprobar que el nombre recibido coincide con el atributo nombre y si es así actualiza el atributo sueldo con el sueldo recibido).

Plantear un segundo método que imprima el nombre y un mensaje si debe o no pagar impuestos (si el sueldo supera a 3000 paga impuestos).

Ejercicio 2

Confeccionar una clase Menu, con los atributos titulo y enlace (ambos son arrays). Crear los métodos necesarios que permitan añadir elementos al menú. Crear los métodos que permitan mostrar el menú en forma horizontal o vertical (según que método llamemos).

Ejercicio 3

Crear una clase cubo que contenga información sobre la capacidad y su contenido actual en litros. Se podrá consultar tanto la capacidad como el contenido en cualquier momento. Dotar a la clase de la capacidad de verter el contenido de un cubo en otro (hay que tener en cuenta si el contenido del cubo origen cabe en el cubo destino, si no cabe, se verterá solo el contenido que quepa). Hacer una página principal para probar el funcionamiento con un par de cubos.

Ejercicio 4

Creamos la clase factura con el atributo de clase IVA (21) y los atributos de instancia ImporteBase, fecha, estado (pagada o pendiente) y productos (array con todos los productos de la factura, que contiene el nombre, precio y cantidad).

Define los métodos AñadeProducto, ImprimeFactura y los getters y setters de los atributos ImporteBase (solo getter, pues su contenido se actualiza automáticamente), fecha y estado.

Ejercicio 5

Crea las clases Animal, Mamifero, Ave, Gato, Perro, Canario, Pinguino y Lagarto. Crea, al menos, tres métodos específicos de cada clase y redefine el/los método/s cuando sea necesario. Prueba las clases en un programa en el que se instancien objetos y se les apliquen métodos. Puedes aprovechar las capacidades que proporciona HTML y CSS para incluir imágenes, sonidos, animaciones, etc. para representar acciones de objetos; por ejemplo, si el canario canta, el perro ladra, o el ave vuela.

Ejercicio 6

Crea la clase Vehiculo, así como las clases Bicicleta y Coche como subclases de la primera. Para la clase Vehiculo, crea los métodos de clase getVehiculosCreados() y getKmTotales(); así como el método de

instancia `getKmRecorridos()`. Crea también algún método específico para cada una de las subclases. Prueba las clases creadas mediante una aplicación que realice, al menos, las siguientes acciones:

- Anda con la bicicleta
- Haz el caballito con la bicicleta
- Anda con el coche
- Quema rueda con el coche
- Ver kilometraje de la bicicleta
- Ver kilometraje del coche
- Ver kilometraje total

Ejercicio 7

Crea la clase `DadoPoker`. Las caras de un dado de poker tienen las siguientes figuras: As, K, Q, J, 7 y 8. Crea el método `tira()` que no hace otra cosa que tirar el dado, es decir, genera un valor aleatorio para el objeto al que se le aplica el método. Crea también el método `nombreFigura()`, que diga cuál es la figura que ha salido en la última tirada del dado en cuestión. Crea, por último, el método `getTiradasTotales()` que debe mostrar el número total de tiradas entre todos los dados. Realiza una aplicación que permita tirar un cubilete con cinco dados de poker.

Ejercicio 8

Queremos gestionar la venta de entradas (no numeradas) del Circo del Sol, que tiene 3 zonas, la zona principal con 1000 entradas disponibles a un precio de 20€, la zona de compra-venta con 200 entradas disponibles a un precio de 35€, y la zona vip con 25 entradas disponibles a un precio de 50€. Define la clase `Zona` con sus atributos y métodos correspondientes, de manera que permita tener información del ingreso total de ventas de todas las entradas (hay que controlar que existen entradas disponibles antes de venderlas), y crea un programa que permita vender las entradas. En la pantalla principal debe aparecer información sobre las entradas disponibles en cada zona, así como su precio y un formulario para vender entradas de la zona seleccionada. Debemos indicar para qué zona queremos las entradas y la cantidad de ellas. Lógicamente, el programa debe controlar que no se puedan vender más entradas de la cuenta.

Ejercicio 9

Diseñar una clase `Coche` con los atributos `matricula`, `modelo` y `precio`. La clase debe tener los atributos de clase `modeloCaro` y `precioCaro`, que contendrá en todo momento el modelo del coche más caro y su precio. Los métodos a incluir son:

- constructor con todos los atributos
- getters y setters
- `toString` (devuelve los datos en columnas de fila de tabla: “matriculamodelo...”).
- Incluir un método de clase “masCaro” que devuelva un `String` con el modelo y el precio del coche más caro.

Diseñar una clase `CocheLujo`, que contendrá todos los atributos y métodos de la clase `Coche` y además un atributo `suplemento` (se pasa en el constructor), que habrá que añadir al precio cuando se consulte a través del método `getPrecio()`, los coches de lujos también hay que tenerlos en cuenta como posible modelo de coche más caro, pero sin contar con el suplemento (solo su precio). En el método `toString` de la clase `CocheLujo` también hay que devolver el suplemento (es una columna más de la fila de tabla devuelta).

Diseñar una página que permita crear coches de la clase Coche y vaya mostrando el listado de los mismos en una tabla, si el coche no es de lujo, en la celda del suplemento mostrará “No procede”. También se mostrará en la cabecera de la página los datos del coche más caro.

Ejercicio 10

Queremos modelar una casa con muchas bombillas, de forma que cada bombilla se puede encender o apagar individualmente. Para ello haremos una clase Bombilla con un atributo privado que almacene si está encendida o apagada, otro para la potencia consumida y por último otro atributo para la ubicación (salón, cocina, etc...); realizar un método que nos diga si una bombilla concreta está encendida, así como los getter y setters necesarios.

Además, queremos poner un interruptor general de la luz, tal que, si saltan los fusibles, todas las bombillas quedan apagadas. Cuando el fusible se repara, las bombillas vuelven a estar encendidas o apagadas, según estuvieran antes del percance.

Diseñar una página que genere las bombillas de una casa y las almacene en un array de sesión. Mostrar las bombillas de manera gráfica (desarrolla tu imaginación) dando la opción de encender y apagar cada una, así como de encender y apagar el interruptor general. Mostrar en todo momento la potencia consumida por las bombillas encendidas.

9.8 Conceptos de la unidad

- La clase se debe crear en un fichero con extensión php, con nombre igual al de la clase.

```
class NombreClase {
    private $atributo1;
    private $atributo2;
    public function __construct($atr1, $atr2) { // método mágico Constructor
        $this->atributo1 = $atr1;
        $this->atributo2 = $atr2;
    }
    public function setAtributo1($atr1) {
        $this->atributo1 = $atr1;
    }
    public function getAtributo1() {
        return $this->atributo1;
    }
    public function nombreFuncion($parametro1, $parametro2, ...) {
        ...
        return valor;
    }
    public function __toString() { // método mágico toString
        return "texto $this->atributo1 texto $this->atributo2";
    }
}
```

- Uso de parámetros opcionales en el constructor.

```
public function __construct($atr1, $atr2, ...){
    $this->$atr1=($atr1==null) ? "valor_por_defecto" : $atr1 ;
    $this->$atr2=($atr2==null) ? "valor_por_defecto" : $atr2 ;
    ...
}
```

Si se omiten algunos valores al crear el objeto, se escribe null en el parámetro omitido, si los valores omitidos están al final, no hace falta poner valor null ni poner comas.

```
$objeto = new ClaseObjeto (valor1, null, valor3, ...)
```

- Uso de una clase almacenada en otro fichero.

```
include_once 'NombreClase.php';
$objeto = new NombreClase(valor1, valor2);
echo $objeto->getAtributo1(); //imprime el atributo1 con llamada al método get
$objeto->setAtributo1(valor); //establece un nuevo valor para el atributo1
$variable=$objeto->nombreFuncion();
echo $objeto; //llama a método toString
var_dump($objeto); //imprime el tipo y valor de todos los atributos
```

- Crear una clase que hereda de otra (normal o abstracta: abstract class)


```
include_once 'ClasePadre.php'; //se deben incluir la clase de la que herede
class ClaseHija extends ClasePadre {
    private $atributo2;
    public function __construct($atr1, $atr2) {
        parent::__construct($atr1); //llama al método constructor de la clase padre
        $this->atributo2=$atr2;
    }
    public function nombreFuncion() { //si el nombre existe en la clase padre, lo redefine
        return parent::metodoClasePadre()."texto"; //llamada a un método padre
    }
}
```

• Atributos y métodos de clase

```
class NombreClase {
    private static $atributoEstatico = valor; // atributo de clase
    public static function nombreFuncionEstatica() { // método de clase
        return nombreClase::$atributoEstatico;
    }

    private $atributo; // atributo de instancia
    public function nombreFuncion($atr){ // método de instancia
        $this->atributo = $atr;
        NombreClase::$atributoEstatico += $atr; // acceso a un atributo estático
    }
}
```

• Como saber a qué clase pertenece un objeto

get_class (objeto): devuelve el nombre de la clase del objeto

Ejemplo:

```
echo get_class ($coche1); //imprime 'Coche'
```

objeto instanceof clase: Devuelve verdadero si el objeto es una instancia de la clase o de una de sus clases hijas. El nombre de la clase debe estar almacenado en una variable, no se puede poner el nombre como texto entre comillas.

Ejemplo:

```
$nombreClase='Coche';
if ($coche1 instanceof $nombreClase){ echo "Es un coche";}
```

is_a (objeto, clase): Devuelve verdadero si el objeto es una instancia de la clase o de una de sus clases hijas. El nombre de la clase si puede ir como un texto entre comillas.

Ejemplo:

```
if (is_a ($coche1, 'Coche')){ echo "Es un coche";}
```

• Aclaraciones

- El acceso a un método de clase (estático) es con '::' en vez de '->' (ejemplo: Coche::kmsTotales)
- Al almacenar objetos en variables de sesión, en la página donde se almacenen o recuperen, la sentencia include de la clase, debe hacerse antes de la sentencia 'session_start()'

- Para evitar errores al recuperar objetos de una sesión (por no hacer el include de la clase antes de session_start o por usar el método __toString en la clase), podemos almacenar los objetos serializados en el array \$_SESSION[] y operar de la siguiente forma:

```
$objeto = unserialize($_SESSION['objeto']); // Obtenemos el objeto unserializando  
$objeto->metodo($parametro); // Hacemos las operaciones necesarias  
$_SESSION['objeto'] = serialize($objeto); // Volvemos a almacenar el objeto  
serializado
```
- Como ya sabemos en php, al recargar la página se pierden todos los datos almacenados en las variables, por tanto, para trabajar con atributos estáticos de clase cuando haya recargas de página, en la clase se usarán variables de sesión para los atributos estáticos en vez de atributos 'private static', ya que de otra forma perderíamos los datos estáticos en cada petición al servidor.

10. Acceso a bases de datos

10.1 Acceso a BBDD desde PHP

Desde una página con código escrito en PHP nos podemos conectar a una base de datos y ejecutar comandos en lenguaje SQL - para hacer consultas, insertar o modificar registros, crear tablas, dar permisos, etc.

PHP puede trabajar con la práctica totalidad de gestores de bases de datos que hay disponibles, tanto comerciales como de código abierto.

Principalmente encontramos tres maneras diferentes de acceder a una base de datos desde PHP:

mysql

La interfaz `mysql` permite acceder a bases de datos MySQL. Esta interfaz se considera obsoleta y se desaconseja su uso en aplicaciones nuevas. No obstante, se ha venido utilizando en multitud de aplicaciones y sería recomendable conocerla.

mysqli

Se trata de una mejora de la interfaz `mysql` (la “i” viene de “improved”). Por ejemplo, la extensión `mysqli` añade ciertas funcionalidades que no tiene `mysql` como las transacciones, los procedimientos almacenados y las sentencias múltiples. En aplicaciones nuevas se recomienda usar esta API, o bien PDO.

PDO

PDO son las siglas de PHP Data Objects (Objetos de Datos de PHP). Proporciona una capa de abstracción de tal forma que los métodos utilizados para acceder a los datos son independientes del sistema gestor de bases de datos utilizado. En la práctica, permite cambiar de SGBD sin cambiar el código PHP. Usaremos este sistema, puesto que será necesario más adelante cuando estudiemos el modelo-vista-controlador, y también usado en frameworks como Laravel.

10.2 PDO (PHP Data Objects)

Establecimiento de una conexión

Para crear una conexión nueva a un servidor, se utiliza el constructor `PDO` de la siguiente manera:

```
$conexion = new PDO("mysql:host=localhost", "root", "root");
```

Observa que, en primer lugar, hay que indicar el gestor de bases de datos que se utilizará. En este caso es `mysql`. Una vez establecida la conexión, cualquier operación que se realice sobre una base de datos se efectuará con métodos que son independientes del SGBD; es decir, no es necesario cambiar el código de la aplicación web, aunque se cambie el SGBD al que se conecta.

```
<!DOCTYPE html>  
<html>
```

```

<head>
  <meta charset="UTF-8">
</head>
<body>
  <?php
    try {
      $conexion = new PDO("mysql:host=localhost;dbname=banco;charset=utf8", "root",
        "root");
      echo "Se ha establecido una conexión con el servidor de bases de datos.";
    } catch (PDOException $e) {
      echo "No se ha podido establecer conexión con el servidor de bases de
        datos.<br>";
      die ("Error: " . $e->getMessage());
    }
  ?>
</body>
</html>

```

Para efectuar una conexión a otro gestor de bases de datos, hay que cambiar `mysql` dentro de la sentencia `$conexion = new PDO("mysql:host=localhost;dbname=banco;charset=utf8", "root", "root");`

Se utilizan las palabras reservadas `pgsql`, `sqlite`, `firebird`, `informix` y `OCI` para establecer conexión con PostgreSQL, SQLite, Firebird, Informix y Oracle respectivamente.

Una vez realizadas todas las operaciones con la BD, debemos cerrar la conexión con la instrucción

```
$conexion = null;
```

Listado completo de una tabla

La extracción de datos de una tabla mediante la interfaz PDO se realiza de forma que los datos de cada fila se transforman en objetos mediante `fetchObject()`.

```

<!DOCTYPE html>
<html>
  </head>
  <body>
    <h2>
      Base de datos <u>banco</u><br>
      Tabla <u>cliente</u><br>
    </h2>
    <?php
      // Conexión a La base de datos
      try {
        $conexion = new PDO("mysql:host=localhost;dbname=banco;charset=utf8", "root", "root");
      } catch (PDOException $e) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
      }
    }
  }

```

```

        die ("Error: " . $e->getMessage());
    }
    $consulta = $conexion->query("SELECT dni, nombre, direccion, telefono FROM cliente");
    ?>
    <table border="1">
    <tr>
        <td><b>DNI</b></td>
        <td><b>Nombre</b></td>
        <td><b>Dirección</b></td>
        <td><b>Teléfono</b></td>
    </tr>
    <?php
        while ($cliente = $consulta->fetchObject()) {
    ?>
        <tr>
            <td><?= $cliente->dni ?></td>
            <td><?= $cliente->nombre ?></td>
            <td><?= $cliente->direccion ?></td>
            <td><?= $cliente->telefono ?></td>
        </tr>
    <?php
    }
    ?>
    </table>
    <br>
    Número de clientes: <?= $consulta->rowCount() ?>
    <?php $conexion=null; ?> //cerramos la conexión a la BD
</body>
</html>

```

10.3 Operaciones sobre una tabla

Las operaciones de inserción, borrado o actualización de datos sobre una tabla se realiza mediante `exec()`.

Como ejemplo, vamos a realizar una inserción mediante el comando `INSERT` en la tabla `cliente` de la base de datos `banco`.

En primer lugar se piden los datos del cliente mediante un formulario como el que se muestra a continuación.

```

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <h1>Base de datos <u>banco</u></h1>
        <h2>Alta de cliente</h2>
        <form action="altaClienteAccion.php" method="post">
            DNI <input type="text" name="dni" required><br>

```

```

Nombre <input type="text" name="nombre"><br>
Dirección <input type="text" name="direccion"><br>
Teléfono <input type="tel" name="telefono"><br>
<input type="submit" value="Enviar">
</form>
</body>
</html>

```

Los datos recogidos en el formulario se envían a `altaClienteAccion.php`.

Antes de realizar la inserción del registro en la base de datos es necesario comprobar que no existe ningún cliente con el DNI del nuevo cliente que se quiere insertar. Una vez comprobado este supuesto, se procede a ejecutar el `INSERT` con los datos del nuevo cliente.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <?php
      // Conexión a la base de datos
      try {
        $conexion = new PDO("mysql:host=localhost;dbname=banco;charset=utf8", "root",
"root");
      } catch (PDOException $e) {
        echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
        die ("Error: " . $e->getMessage());
      }
      // Comprueba si ya existe un cliente con el DNI introducido
      $consulta = $conexion->query("SELECT dni FROM cliente WHERE dni=".$_POST['dni']);
      if ($consulta->rowCount() > 0) {
        ?>
        Ya existe un cliente con DNI <?= $_POST['dni'] ?><br>
        Por favor, vuelva a la <a href="altaClienteFormulario.php">página de alta de
        cliente</a>.

        <?php
        } else {
          $insercion = "INSERT INTO cliente (dni, nombre, direccion, telefono) VALUES
('".$_POST[dni]."', '".$_POST[nombre]."', '".$_POST[direccion]."', '".$_POST[telefono]."'");
          //echo $insercion;
          $conexion->exec($insercion); echo "Cliente dado de alta correctamente.";
          $conexion=null;
        }
        ?>
      </body>
</html>

```

10.4 Error al iniciar servicio MYSQL

A veces, alguna base de datos puede corromperse por alguna operación no terminada, y esto provoca que no pueda iniciarse el servicio MYSQL en Xampp. Para corregirlo podemos seguir los siguientes pasos:

Realizar todas las operaciones dentro de la carpeta '**C:\xampp\mysql**'















- Cambiar el nombre de la carpeta data por '**data_old**'
- Crear una nueva carpeta llamada '**data**'
- Volcar el contenido de la carpeta '**backup**' dentro de la nueva carpeta '**data**'
- Copiar el archivo '**ibdata1**' que está dentro de la carpeta '**data_old**' dentro de la nueva carpeta '**data**' sobrescribiendo.
- Y por último ir restaurar las bases de datos que nos hagan falta, copiándolas desde la carpeta '**data_old**' a '**data**', teniendo en cuenta que, si restauramos la que estaba corrupta, tendremos el mismo problema de nuevo.

10.5 Ejercicios para resolver

Ejercicio 1

Crea una aplicación web que permita hacer listado, alta, baja y modificación sobre la tabla `cliente` de la base de datos `banco`.

- Para realizar el listado bastará un `SELECT`, tal y como se ha visto en los ejemplos.
- El alta se realizará mediante un formulario donde se especificarán todos los campos del nuevo registro. Luego esos datos se enviarán a una página que ejecutará `INSERT`.
- Para realizar una baja, se mostrará un botón que ejecutará `DELETE`.
- La modificación se realiza mediante `UPDATE`.

Mantemiento de clientes					
DNI	Nombre	Dirección	Teléfono		
3534534	Cacerolo Tontoñez	Almogía	123456	 Eliminar	 Modificar
45678	Mota	Calle Falsa, 123	555 444333	 Eliminar	 Modificar
456958	Javier Roviralta	Calle Olvido, 77	555 76845	 Eliminar	 Modificar
555	Luis José	Montserrat Roig, 10	5555 234233	 Eliminar	 Modificar
657456	uytutyut	ghjfhgj	67867	 Eliminar	 Modificar
65767	Pepito Lupiañez	Alhaurín	867867867	 Eliminar	 Modificar
76859	Ignacio	Periquito, 333	555 325476	 Eliminar	 Modificar
789654	Yren	Calle Verdadera, 98	555 98765	 Eliminar	 Modificar
873475933	Maria Sol	Calle Flor	555 123456	 Eliminar	 Modificar
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	 Nuevo cliente	

Ejercicio 2

Modifica el programa anterior añadiendo las siguientes mejoras:

- El listado debe aparecer paginado en caso de que haya muchos clientes.
- Al hacer un alta, se debe comprobar que no exista ningún cliente con el DNI introducido en el formulario.
- La opción de borrado debe pedir confirmación.
- Cuando se realice la modificación de los datos de un cliente, los campos que no se han cambiado deberán permanecer inalterados en la base de datos.

Ejercicio 3

Modifica la tienda virtual realizada en el ejercicio 5 de las plumas, del tema 7, de tal forma que todos los datos de los artículos se obtengan de una base de datos. Crea la base de datos plumas y la tabla productos, donde se almacenan las plumas/bolígrafos, incluyendo una columna detalle. Añade un botón ‘Administrar productos’ en la página principal, que lleve a una segunda página para el mantenimiento de los productos en la tabla (alta, baja y modificación).

Ejercicio 4

Crea el programa GESTISIMAL (GESTIÓN SIMplificada de ALmacén) para llevar el control de los artículos de un almacén. De cada artículo se debe saber el código, la descripción, el precio de compra, el precio de venta y el stock (número de unidades). La entrada y salida de mercancía supone respectivamente el incremento y decremento de stock de un determinado artículo. Hay que controlar que no se pueda sacar más mercancía de la que hay en el almacén. El programa debe tener, al menos, las siguientes funcionalidades: listado, alta, baja, modificación, entrada de mercancía y salida de mercancía.

GESTISIMAL

Código	Descripción	Precio de compra	Precio de venta	Margen	Stock				
h020	Barra acero 16mm. longitud 6m.	35.30	45.40	10.1	50	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
h007	barra para cortina 2,00 m.	10.30	22.33	12.03	5	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
h005	Caja tuercas 16mm.	21.00	25.05	4.05	20	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
h006	chapa galvanizada	10.50	20.55	10.05	3	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>
m001	Estantería para pared.	25.30	30.60	5.3	5	<div>Eliminar</div>	<div>Modificar</div>	<div>Entrada</div>	<div>Salida</div>

Página 1 de 3

Primera

Anterior

Siguiente

Última

Código	Descripción	Precio de compra	Precio de venta	Stock	
<div></div>	<div></div>	<div></div>	<div></div>	<div></div>	<div>Nuevo artículo</div>

Ejercicio 5

Modifica el programa anterior añadiendo las siguientes mejoras:

- Comprueba la existencia del código en el alta, la baja y la modificación de artículos para evitar errores.
- Añade un botón comprar para añadir el código del producto a una cesta o carrito almacenado en una sesión.
- Añade un botón ‘procesar pedido’ para crear un fichero de factura de la venta, donde se incluya un listado con los datos de los productos vendidos (los que estaban en la cesta) mostrando su precio con iva de 21% añadido. También se incluirá al final del fichero el total de la factura con iva. Mostrar un enlace al fichero generado para consultar la factura.

Ejercicio 6

Diseñar una aplicación web para consultar y actualizar tu horario de clase. Nada más cargar la página debe mostrar una tabla con tu horario semanal recuperado de la base de datos. Cada asignatura mostrada en la tabla es realmente un botón y al pulsar el botón de cada asignatura abre otra página que contiene un formulario con

un desplegable que incluye todas las asignaturas posibles, y al elegir una y darle a grabar, actualizará el horario y volverá a la página principal.

Crear una base de datos llamada escuela y una tabla llamada horario con los siguientes campos: 'dia', 'primera', 'segunda', 'tercera', 'cuarta', 'quinta', 'sexta'. Los campos se corresponden con la información del día de la semana, y sus seis horas de clase, por lo que en la tabla habrá una fila por cada día de la semana. Todos los campos son de tipo varchar y la clave de la tabla es 'dia', ya que no se puede repetir.

Ejercicio 7

Modifica el ejercicio del carrito de la compra que obtenía los productos de un fichero, para que ahora estén almacenados en una base de datos llamada tienda. Por tanto, la aplicación deberá recuperar todos los productos de la base de datos al inicio. Debe añadirse la funcionalidad que permita añadir nuevos productos en la Base de Datos, así como eliminarlos. Incluir un “type file” que permita subir al servidor, el archivo de imagen correspondiente al producto nuevo que se está añadiendo.

10.6 Conceptos de la unidad

Crear conexión a la BD

```
try {
    $conexion = new PDO("mysql:host=localhost;dbname=nombreBD;charset=utf8",
"usuarioBD", "contraseñaBD");
    echo "Se ha establecido una conexión con el servidor de bases de datos.";
} catch (PDOException $e) {
    echo "No se ha podido establecer conexión con el servidor de bases de
datos.<br>";
    die ("Error: ". $e->getMessage());
}
```

Recorrido de una tabla

```
$consulta = $conexion->query("SELECT dni FROM cliente");
while ($cliente = $consulta->fetchObject()) {
    echo 'br'. $cliente->dni;
}
```

Consulta del número de filas de una consulta

```
echo $consulta->rowCount();
```

Cerrar conexión a la BD

```
$conexion=null; ($conexion->close() de versiones anteriores no funciona)
```

Insertar, modificar o borrar un registro

```
$insercion = "INSERT INTO cliente (dni, nombre, direccion, telefono) VALUES
('$ _POST[dni]', '$ _POST[nombre]', '$ _POST[direccion]', '$ _POST[telefono]')";
$delete = "DELETE FROM cliente WHERE dni='$ _POST[dni]'";
$update= "UPDATE cliente SET  nombre=\"$ _POST[nombre]\",
direccion=\"$ _POST[direccion]\", telefono=\"$ _POST[telefono]\" WHERE
dni=\"$ _POST[dni]\"";
```

```
$conexion->exec($insercion);
```

Funciones SQL

LIMIT valor1, valor2 => al final de un SELECT filtra filas del resultado de la consulta a partir de la fila número valor1 hasta un número de filas igual a valor2

now() => usado como valor en un insert o update, toma la fecha/hora del sistema

Tipo fecha (date, time, datetime, timestamp)

El formato con el que se guardan las tipos fecha en mysql es: date('Y-m-d H:i:s')

Cuando se inserta y se recupera el valor de una fecha/hora de la base de datos, hay que hacerlo siempre como un string con el formato anterior, de modo que si después de recuperarla quiero formatearla de nuevo con un formato distinto, debo convertirla primero a entero con strtotime.

11. Modelo Vista Controlador

El Modelo Vista Controlador (Model View Controller) - también conocido por sus siglas MVC - es un patrón de diseño de software que separa una aplicación en tres grandes bloques: el acceso a datos, la interfaz de usuario y la lógica de negocio.

La manera de implementar este patrón no es única. En este libro mostramos una de las muchas aproximaciones posibles, pero el lector podrá encontrar implementaciones diferentes en otros manuales.

Para ilustrar los conceptos del patrón MVC vamos a trabajar con un ejemplo muy simple: un gestor de ofertas de una pizzería. Los datos sobre las ofertas se guardan en una tabla llamada `oferta` dentro de la base de datos `pizzeria`. El fichero `pizzeria.sql` se encuentra disponible en [GitHub](#)¹. Cada oferta tiene un identificador único que es un número entero. El campo correspondiente en la base de datos es un entero “autoincrementable”, por tanto, el propio MySQL se encarga de asignar un valor cuando se crea una nueva oferta. Para cada oferta hay también un título, una imagen (el nombre del fichero de imagen) y una descripción.

La estructura de la aplicación quedaría de la siguiente manera:

```
.
├── Controller
│   ├── actualizaOferta.php
│   ├── borraOferta.php
│   ├── grabaOferta.php
│   ├── index.php
│   ├── nuevaOferta.php
│   └── updateOferta.php
├── index.php
├── Model
│   ├── Oferta.php
│   └── PizzeriaDB.php
├── View
│   ├── nuevaOferta_view.php
│   ├── index_view.php
│   ├── actualizaOferta_view.php
│   └── images
│       ├── pizza1.jpg
│       ├── pizza2.jpg
│       └── pizza3.jpg
```

11.1 El modelo

En una aplicación web, los datos se guardan normalmente en una base de datos. La parte del modelo dentro del MVC será, por tanto, una capa que da acceso a los datos y que abstrae mediante una clase todas las consultas, inserciones, borrados, etc. que se realizan en las tablas mediante SQL.

Por ejemplo, si una aplicación gestiona artículos de un almacén, deberá tener una tabla - con nombre `articulo` - en la base de datos que almacene los datos de los artículos: código, descripción, tipo, precio, etc. En este caso, como parte del modelo habrá una clase con nombre `Articulo.php` donde estará implementado el acceso a los datos de los artículos para sacar listados, dar de alta nuevos artículos, modificar artículos existentes, etc.

Un ejemplo de oferta podría ser la que tiene como identificador el 3, como título “Bebida gratis pidiendo dos pizzas”, como imagen “pizza3.jpg” y como descripción algún texto más largo como “Pidiendo dos pizzas de cualquier tipo te regalamos dos bebidas (no incluye bebidas alcohólicas de alta graduación)”.

El modelo estará implementado dentro de la carpeta `Model`. Esta carpeta contendrá todo el código que tenga que ver con el acceso a los datos.

El fichero que se encarga del establecimiento de la conexión con la base de datos es `PizzeriaDB.php` y se muestra a continuación.

```
abstract class PizzeriaDB {
    private static $server = 'localhost';
    private static $db = 'pizzeria';
    private static $user = 'root';
    private static $password = '';

    public static function connectDB() {
        try {
            $connection = new PDO("mysql:host=".self::$server.";dbname=".self::$db.";charset=utf8",
self::$user, self::$password);
        } catch (PDOException $e) {
            echo "No se ha podido establecer conexión con el servidor de bases de datos.<br>";
            die ("Error: " . $e->getMessage());
        }

        return $connection;
    }
}
```

A partir de este momento, cada vez que queramos establecer una conexión con la base de datos bastará con escribir `$miConexion = PizzeriaDB::connectDB()`.

Si en el futuro se cambia el nombre de usuario de la base de datos, la contraseña o algún otro dato de la conexión, únicamente habría que actualizar el fichero `PizzeriaDB.php`; el resto de la aplicación permanecería intacta.

A continuación, definiremos la clase `Oferta` en base a la tabla `oferta` de la base de datos. Los atributos de instancia deben tener los mismos nombres que los campos de la tabla.

En la clase `Oferta` se define el constructor y los setter.

El método `insert()` graba los datos de un objeto de la clase `Oferta` en la base de datos. El método `delete()` aplicado a un objeto, mira el identificador de la oferta que tiene ese objeto y borra en la base de datos el registro que tiene ese identificador. Y el método `update()` actualiza los campos en la base de datos con el valor de los atributos del objeto, para aquella fila de la tabla con el mismo identificador.

El método `getOfertas()` devuelve un array de objetos con toda la información de las ofertas disponibles.

Por último, el método `getOfertaById()` devuelve un objeto con la información de la oferta que tiene un determinado identificador. Obviamente, los datos de esa oferta se han tenido que buscar en la base de datos.

El código de `Oferta.php` se muestra a continuación.

```
require_once 'PizzeriaDB.php';

class Oferta {
    private $id;
    private $titulo;
    private $imagen;
    private $descripcion;

    function __construct($id=0, $titulo="", $imagen="", $descripcion="") {
        $this->id = $id;
        $this->titulo = $titulo;
        $this->imagen = $imagen;
        $this->descripcion = $descripcion;
    }

    public function getId() {
        return $this->id;
    }

    public function getTitulo() {
        return $this->titulo;
    }

    public function getImagen() {
        return $this->imagen;
    }

    public function getDescripcion() {
        return $this->descripcion;
    }

    public function insert() {
        $conexion = PizzeriaDB::connectDB();
        $insercion = "INSERT INTO oferta (titulo, imagen, descripcion) VALUES ('$this->titulo',
'$this->imagen', '$this->descripcion')";
        $conexion->exec($insercion);
        $conexion=null;
    }
}
```

```

public function delete() {
    $conexion = PizzeriaDB::connectDB();
    $borrado = "DELETE FROM oferta WHERE id='$this->id'";
    $conexion->exec($borrado);
    $conexion=null;
}

public function update() {
    $conexion = PizzeriaDB::connectDB();
    $actualiza = "UPDATE oferta SET titulo='$this->titulo', imagen='$this->imagen',
        descripcion='$this->descripcion'
        WHERE id='$this->id'";
    $conexion->exec($actualiza);
    $conexion=null;
}

public static function getOfertas() {
    $conexion = PizzeriaDB::connectDB();
    $seleccion = "SELECT id, titulo, imagen, descripcion FROM oferta";
    $consulta = $conexion->query($seleccion);

    $ofertas = [];

    while ($registro = $consulta->fetchObject()) {
        $ofertas[] = new Oferta($registro->id, $registro->titulo, $registro->imagen, $registro-
>descripcion);
    }

    $conexion=null;
    return $ofertas;
}

public static function getOfertaById($id) {
    $conexion = PizzeriaDB::connectDB();
    $seleccion = "SELECT id, titulo, imagen, descripcion FROM oferta WHERE id=$id";
    $consulta = $conexion->query($seleccion);
    if ($consulta->rowCount()>0) {
        $registro = $consulta->fetchObject();
        $oferta = new Oferta($registro->id, $registro->titulo, $registro->imagen, $registro-
>descripcion);
        $conexion=null;
        return $oferta;
    }else{
        $conexion=null;
        return false;
    }
}
}

```

El objetivo de crear `Oferta.php` no es otro que abstraer el acceso a la base de datos. A partir de este momento ya no tendremos que lidiar directamente con las conexiones, los `SELECT`, `INSERT`, etc. ya que eso lo hace ahora la clase `Oferta`.

11.2 La vista

En una aplicación realizada mediante el patrón MVC, la vista corresponde a la interfaz de usuario. La vista tiene código HTML y unos “huecos” donde se colocará la información que pasará el controlador. Normalmente la vista - al ser la interfaz de usuario - contiene CSS, Javascript, JQuery, etc.

Todo lo que tenga que ver con la vista se coloca en la carpeta `View`. Normalmente las imágenes y otros ficheros que necesita la aplicación se colocan aparte, en una carpeta con nombre `Assets` (activos). Nosotros para simplificar colocaremos las imágenes dentro de la vista, dentro de la carpeta `images`.

Los datos que le pasa el controlador a la vista van en un array asociativo llamado `$data[]`. Si el controlador y la vista lo hacen personas diferentes, se tendrán que poner de acuerdo en los nombres de los datos que se pasan de un sitio a otro. Por ejemplo, el programador que implementa el controlador le puede decir al diseñador que se encarga de la interfaz que el nombre del usuario logueado es `$data['usuario']`; de esta forma, el diseñador podría colocar en la interfaz el nombre de usuario escribiendo por ejemplo:

```
<h3>Usuario: <?= $data['usuario'] ?></h3>
```

A continuación, se muestra el código del fichero `index_view.php` que es la vista encargada de mostrar todas las ofertas.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Listado de ofertas</title>
  </head>
  <body>
    <h1>Pizzeria Peachepe</h1>
    <a href="../Controller/nuevaOferta.php">Nueva oferta</a>
    <hr>
    <?php
      foreach($data['ofertas'] as $oferta) {
        ?>
        <h3><?=$oferta->getTitulo()?></h3>
        <br>
        <p><?=$oferta->getDescripcion()?></p><br>
        <a href="../Controller/borraOferta.php?id=<?=$oferta->getId()?>">Borrar</a>
      <?php
      }
    ?>
  </body>
</html>
```

Otro fichero de vista - `nuevaOferta_view.php` - es el que muestra un formulario para introducir los datos de las nuevas ofertas. El código de este fichero se muestra a continuación.


```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <form action="../Controller/grabaOferta.php" enctype="multipart/form-data"
    method="POST">
      <h3>Título</h3>
      <input type="text" size="40" name="titulo">
      <h3>Imagen</h3>
      <input type="file" id="imagen" name="imagen">
      <br><h3>Descripción</h3>
      <textarea name="descripcion" cols="60" rows="6">
      </textarea><hr>
      <input type="submit" value="Aceptar">
    </form>
  </body>
</html>

```

Otro fichero de vista – `actualizaOferta_view.php` - es el que muestra un formulario con los datos de la oferta seleccionada, para modificar los datos deseados. El código de este fichero se muestra a continuación.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title></title>
  </head>
  <body>
    <form action="../Controller/updateOferta.php" enctype="multipart/form-data"
    method="POST">
      <input type="hidden" name="id" value="<?= $data['oferta']->getId() ?>">
      <h3>Título</h3>
      <input type="text" size="40" name="titulo" value="<?= $data['oferta']->getTitulo() ?>">
      <h3>Imagen</h3>
      <input type="hidden" name="imagenAnterior" value="<?= $data['oferta']->getImagen() ?>">
      <br>
      <input type="file" id="imagen" name="imagen">
      <br><h3>Descripción</h3>
      <textarea name="descripcion" cols="60" rows="6">
      <?= $data['oferta']->getDescripcion() ?>
      </textarea><hr>
      <input type="submit" value="Aceptar">
    </form>
  </body>
</html>

```

11.3 El controlador

El controlador contiene lo que se da en llamar la lógica de negocio de la aplicación. Por ejemplo, en una aplicación de gestión de nóminas, el cálculo del sueldo bruto a partir de las horas trabajadas formaría parte de esa lógica de negocio. También es el controlador el encargado de juntar todas las piezas haciendo uso del modelo y dando la orden de cargar las vistas.

El fichero `index.php` del controlador obtiene todos los datos de las ofertas y se los envía a la vista `listado.php`.

```
<?php require_once '../Model/Oferta.php';  
    // Obtiene todas las ofertas  
    $data['ofertas'] = Oferta::getOfertas();  
    // Carga la vista de listado  
    include '../View/index_view.php';
```

El fichero `nuevaOferta.php` simplemente da la orden de cargar la vista que contiene el formulario donde se introducirán los datos de la nueva oferta.

```
<?php  
    // Carga la vista del formulario de alta de oferta  
  
    include '../View/nuevaOferta_view.php';
```

El fichero `grabaOferta.php` recoge los datos enviados por el formulario y hace uso del modelo para grabar la información en la base de datos.

```
<?php  
    require_once '../Model/Oferta.php';  
    // sube la imagen al servidor  
    move_uploaded_file($_FILES["imagen"]["tmp_name"], "../View/images/" .  
        $_FILES["imagen"]["name"]);  
    // inserta la oferta en la base de datos  
    $ofertaAux = new Oferta("", $_POST['titulo'], $_FILES["imagen"]["name"],  
        $_POST['descripcion']);  
    $ofertaAux->insert();  
    header("Location: ../Controller/index.php");
```

El fichero `actualizaOferta.php` recoge los datos de la oferta de la BD a partir del id recibido, para que estén disponibles en la vista que incluye el formulario que permite modificar los datos de la oferta.

```
<?php  
    require_once '../Model/Oferta.php';  
    $data['oferta']=Oferta::getOfertaById($_REQUEST['id']);
```

```
// Carga la vista del formulario de alta de oferta
include '../View/actualizaOferta_view.php';
```

El fichero `updateOferta.php` recoge los datos enviados por el formulario y hace uso del modelo para actualizar la información de la oferta recibida en la base de datos.

```
<?php
require_once '../Model/Oferta.php';

$oferta=Oferta::getOfertaById($_REQUEST['id']);
$imagen=$_REQUEST['imagenAnterior'];
if ($_FILES["imagen"]["name"]!="") {
    // sube la imagen al servidor
    move_uploaded_file($_FILES["imagen"]["tmp_name"], "../View/images/" .
$_FILES["imagen"]["name"]);
    $imagen=$_FILES["imagen"]["name"];
}

// actualiza la oferta en la base de datos
$ofertaAux = new Oferta($_REQUEST['id'], $_POST['titulo'], $imagen, $_POST['descripcion']);
$ofertaAux->update();
header("Location: ../Controller/index.php");
```

Por último, el fichero `borraOferta.php` elimina de la base de datos la oferta seleccionada.

```
<?php
require_once '../Model/Oferta.php';
$ofertaAux = new Oferta($_GET['id']);
$ofertaAux->delete();
header("Location: ../Controller/index.php");
```

11.4 Ejercicios para resolver

Ejercicio 1

Crea un blog siguiendo las pautas que se marcan a continuación:

- En un blog hay como mínimo una cabecera, una serie de artículos y un pie de página.
- Los artículos se almacenan en una base de datos. Sobre cada artículo se debe saber al menos el título, la fecha de publicación (o fecha y hora) y el contenido. Además cada artículo tendrá un identificador o código único (puede ser un código que se auto-incremente).
- El identificador puede ser un número que se vaya incrementando él solo.
- La fecha se puede coger del sistema cuando se graba un nuevo artículo.
- Crea la clase `BlogDB` para aislar los datos de la conexión a la base de datos donde se guardan los artículos.
- Crea la clase `Articulo` con los mismos atributos que campos hay en la tabla `articulo` de la base de datos. Esta clase debe tener implementado el constructor y opcionalmente los getter y setter (se pueden crear de forma automática con `Alt + Insert`).

- g) La clase `Articulo` tendrá también los métodos `insert` y `delete`, que deben insertar y borrar respectivamente un artículo de la base de datos.
- h) La clase `Articulo` debe tener también un método de clase `getArticulos()` que devuelva en un array todos los artículos de la base de datos.

Ejercicio 2

Mejora el blog de tal forma que se permita la modificación de un artículo y añada estilos a la aplicación para hacerla más atractiva

Ejercicio 3

Realizar la aplicación anterior del carrito de la compra en su última versión, pero esta vez cumpliendo el patrón modelo-vista-controlador y los productos serán almacenados en la tabla con los siguientes atributos: código (autonumérico), nombre, precio y stock, con los siguientes métodos en la clase: constructor, reponer, vender y los getters de los atributos. Se mantienen las siguientes comprobaciones: al añadir al carrito de la compra, se comprobará si hay existencias del producto (no solo hay que comprobar las existencias sino también la cantidad que hay en la cesta) y si no hay existencias no añadir el producto a la cesta. También se mantiene el botón “Comprar” en la cesta, que la pulsar dejará la cesta vacía y restará las unidades vendidas al stock de cada producto. También puedes agregar confirmación antes de borrar un producto, y generar la factura o ticket en un fichero y abrirlo en otra pestaña al finalizar la compra.

Ejercicio 4

Realizar una aplicación web en php siguiendo el MVC. La aplicación muestra en su página principal un listado de alumnos (matricula, nombre, apellidos, curso). Debe existir un botón “Añadir Alumno” para dirigirnos a un formulario de alta de un nuevo alumno. También debe haber un botón “Asignaturas” que dirija hacia otra página con el listado de asignaturas (código autonumérico y nombre) y que ofrezca la posibilidad de añadir y eliminar asignaturas.

En la página principal junto a cada alumno debe aparecer un botón detalles, que abrirá una página con las asignaturas en las que está matriculado. Junto a cada una de ellas debe aparecer un botón para desmatricular al alumno de esa asignatura. También habrá un formulario con un desplegable que incluya todas las asignaturas disponibles para poder matricular al alumno en una nueva asignatura.

Crear una BD “escuela” que permita almacenar la información de la aplicación con las tablas alumno (matricula, nombre, apellidos, curso), asignatura (código autonumérico y nombre) y alumno-asignatura (matricula y código-asignatura).

Ejercicio 5

Siguiendo el patrón MVC, realizar la aplicación anterior del carrito de la compra el Ejercicio 3, pero añadiendo un control de acceso a la tienda a través de usuarios registrados. Tendremos 2 perfiles de usuarios ‘admin’ y ‘cliente’.

El usuario administrador no tendrá opción de comprar, solo de administrar los productos de la tienda y el stock de los mismo (añadir y eliminar productos y añadir stock de los mismos).

El usuario cliente solo tendrá la posibilidad de comprar productos. Una vez iniciada sesión con un usuario cliente, los productos añadidos a la cesta no serán guardados en una sesión, sino en una tabla de la base de datos, de manera que cuando el usuario se vuelva a conectar en cualquier otro dispositivo siga teniendo los productos en la cesta.

Para ello tendrás que usar vistas distintas según el perfil del usuario. Es necesario añadir 2 tablas a la base de datos: 'usuario' y 'cesta'. La tabla usuario tendrá las columnas 'id' (autonumérico), 'nombre', 'pass' y 'rol' (admin o cliente), y la tabla cesta tendrá las columnas 'id_cliente', 'cod_producto' y 'cantidad' (tendrá una fila por cada producto en la cesta de cada usuario, con su cantidad correspondiente). En la página principal será necesario hacer login para acceder a la aplicación.

1.5 Conceptos de la unidad

Modelo.

- Carpeta 'Model'
- Una clase con el nombre de la base de datos 'nombreDB', que devuelve la conexión a la base de datos a través del método estático 'connectDB()'
- Una clase por cada tabla con los atributos de instancia correspondientes a las columnas de la tabla, y los siguientes métodos:
 - Métodos de instancia: insert(), delete(), update() y los que hagan falta
 - Métodos de clase: getElementos() que devuelve un array con los objetos correspondientes a todas las filas de la tabla y getElementoById(\$id) que devuelve el objeto correspondiente a la fila de la tabla con el id pasado por parámetro

Vista.

- Carpeta 'View'
- Son las encargadas de generar toda la interfaz html que presenta la página, despreocupándose de la lógica que recupera o genera los datos.
- Los datos que utilice la vista son recogidos del array asociativo \$data[]
- Las vistas nunca son enlazadas, sino que son incluidas en los controladores, por tanto los enlaces de los atributos html 'href' y 'action', siempre apuntan a los controladores, incluso si el único código que tiene dicho controlador es el include de una vista.
- Los archivos relacionados con la interfaz van en esta carpeta, tales como css, imágenes, etc, en la subcarpeta correspondiente
- Como regla general para mayor claridad del código, asignaremos como nombre de la vista, el mismo nombre del controlador que la incluye más la coletilla '_view', aunque si una misma vista es incluida en más de un controlador, habrá que asignar un nombre alternativo.

Controladores

- Carpeta Controller
- Esta parte es la encargada de coordinar todo el funcionamiento de la aplicación realizando las llamadas a los métodos de las clases ubicadas en el modelo, para preparar los datos que van a necesitar las vistas almacenadas en la carpeta View

- En los controladores está toda la lógica de la aplicación, tales como acceso a BD o ficheros, control de sesiones, cookies, etc...
- Deben tener la sentencia include de las clases definidas en el Modelo, para poder operar con las BD a través de las clases de la carpeta Model
- En la raíz se suele ubicar un archivo 'index.php' con la única finalidad de redirigir al 'index.php' ubicado en la carpeta controller, que es el archivo principal de la aplicación

12. Servicios web (web services)

12.1 ¿Qué son los servicios web?

En la actualidad, desarrollar una aplicación no implica necesariamente implementar todas y cada una de las características y funcionalidades desde cero. Hay muchos servicios disponibles tanto gratuitos como de pago que proporcionan datos que se pueden reutilizar en una nueva aplicación.

Por ejemplo, imagina que estás desarrollando una aplicación en la que quieres incluir, entre otras cosas, los resultados de los partidos de la liga de fútbol. Una opción sería tener a alguien pendiente de las retransmisiones de los partidos por la radio o la televisión. Esta persona tendría que ir actualizando manualmente la información en una base de datos creada al efecto.

Ya existen empresas que ofrecen servicios de resultados deportivos en tiempo real; por tanto, lo más sensato sería comprar estos servicios para incluir la información en nuestra aplicación.

El funcionamiento sería fácil. Nuestra aplicación pediría información a la aplicación que proporciona los servicios web de la forma “Dame el resultado del partido del equipo tal con el equipo cual” o “Dame un listado con los diez equipos que van primeros en la clasificación”.

De una forma muy coloquial, podríamos decir que un servicio web es un mecanismo mediante el cual dos aplicaciones web se pueden comunicar y se pueden transmitir datos. Normalmente una de las aplicaciones es la proveedora de servicios y la otra aplicación es la consumidora de servicios. En este capítulo veremos cómo proveer y consumir servicios web con PHP.

Los servicios web se pueden implementar de diferentes maneras. Dos de las más usadas son SOAP (Simple Object Access Protocol) y REST (Representational State Transfer).

SOAP utiliza exclusivamente el formato XML para la transmisión de datos por lo que suele emplear mucho ancho de banda. En las comunicaciones se intercambian “objetos SOAP” que deben estar bien definidos previamente. Es típico que cuando se utiliza SOAP exista un contrato formal entre el cliente y el servidor (entre el consumidor y el proveedor). La utilización de SOAP está muy extendida en servicios financieros, pasarelas de pago y servicios de telecomunicaciones (p. ej. plataformas de envío de SMS).

REST puede trabajar tanto con XML como con JSON. Las peticiones se pueden realizar mediante una simple URL. REST se utiliza mucho en redes sociales y para proveer servicios a móviles. Google, Facebook, Twitter, LinkedIn y Tripadvisor son ejemplos de empresas que proveen servicios REST. Para consumir un servicio web basado en REST sólo se requiere conocer su URL aunque en la práctica se utilizan varias formas de modificación del pedido, como el envío de header para solicitar operaciones sobre los datos.

12.2 Consumo de servicios REST

La manera en que se consumen (se usan) los servicios web puede variar según el proveedor. Es recomendable leer bien la documentación del servicio y prestar atención a los ejemplos.

La comunicación entre cliente y servidor es sencilla, a diferencia de lo que es un Webservice SOAP, se trata simplemente de enviar un pedido HTTP y procesar su respuesta. Existen varias opciones disponibles, veamos las más comúnmente utilizadas:

- cURL

La librería cURL permite realizar peticiones a servidores remotos. Tiene sus pros y sus contras, pero es un método de bastante bajo nivel, por lo que nosotros optaremos por la siguiente alternativa.

- file_get_contents

Una opción alternativa (y generalmente más conveniente) es el uso de la función file_get_contents. Esta función recibe como parámetro el nombre del archivo que se quiere leer y devuelve todo su contenido como un string. Ejemplo:

```
$resultado = file_get_contents("http://miescuela.com/alumnos");
```

Los resultados se procesan de acuerdo al formato de la respuesta, siendo las más comunes en formato XML o JSON. En el caso de tratarse de un XML, lo mejor es usar la biblioteca SimpleXML, y en el caso de json que es el que usaremos nosotros, con la función json_decode será suficiente.

Existen servicios web tanto de pago como gratuitos. Para usar estos servicios, casi siempre es necesario registrarse y, en muchas ocasiones, obtener una clave o token (API key) para usar en cada consulta al servicio, así de esta manera se puede tener un control de las veces que hace uso del servicio cada usuario, incluso limitar el número de consultas por periodo de tiempo, por ejemplo, 500 consultas por hora o día.

Como ejemplo, vamos a usar los servicios de OpenWeatherMap, una web que nos proporciona los datos meteorológicos de más de 200.000 ciudades.

La página principal de OpenWeatherMap (figura 12.2.1) es <http://openweathermap.org/>.

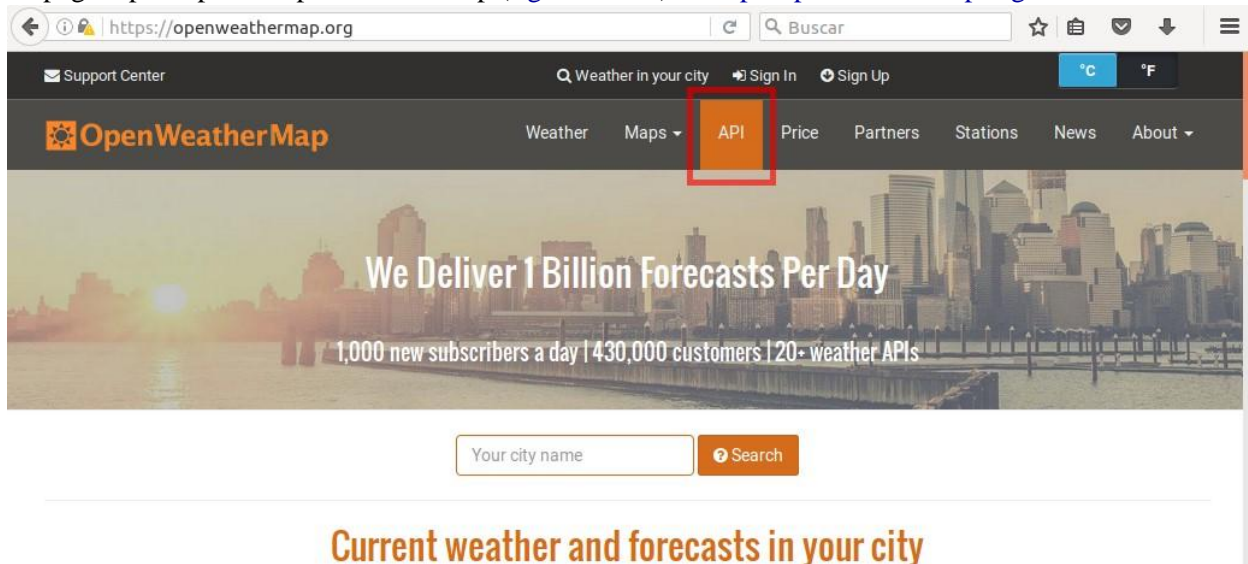


Figura 12.2.1: Página principal de OpenWeatherMap

Desde la página principal se puede acceder a todo lo concerniente a la API REST haciendo click en API

(figura 12.2.1).

The screenshot shows the OpenWeatherMap website with a dark header containing the logo and navigation links: Weather, Maps, API, Price, Partners, Stations, News, and About. Below the header, a text block states: "Our weather API is simple, clear and free. We also offer higher levels of support, please see our **paid plan options**. To access the API you need to sign up for an **API key** if you are on a free or paid plan." Below this, three service categories are listed: "Current weather data", "5 day / 3 hour forecast", and "16 day / daily forecast". Each category has a "Subscribe" button highlighted with a red box. The "Current weather data" section lists: "Access current weather data for any location including over 200,000 cities" and "Current weather is frequently updated". The "5 day / 3 hour forecast" section lists: "5 day forecast is available at any location or city" and "5 day forecast includes weather data". The "16 day / daily forecast" section lists: "16 day forecast is available at any location or city" and "16 day forecasts includes daily weather".

Figura 12.2.2: Suscripción a los servicios de OpenWeatherMap

Para usar los servicios web de esta página es obligatorio registrarse y obtener una clave única. Mediante esta clave, OpenWeatherMap sabe quién accede y cuántas veces lo hace. Hay un número limitado de accesos por día de forma gratuita.

En este manual ilustraremos ejemplos que muestran el tiempo actual. Para probar estos ejemplos es necesario registrarte; para ello, haz click en el botón Subscribe del apartado Current weather data (figura 12.2.2).

Haz click en Get API key and Start como se indica en la figura 12.2.3.

Current weather and forecasts collection

	Free	Startup	Developer	Professional	Enterprise
Price <small>Price is fixed, no other hidden costs.</small>	Free	40 USD / month	180 USD / month	470 USD / month	2,000 USD / month
Subscribe	Get API key and Start	Subscribe	Subscribe	Subscribe	Subscribe
Calls per minute (no more than)	60	600	3,000	30,000	200,000

Figura 12.2.3: Obtención de la clave API A continuación, haz click en Sign up para registrarte figura 12.2.4.

How to start

Home / API / How to start

To get access to weather API you need an API key whatever account you choose from Free to Enterprise.

How to get API key (APPID)

Sign up

to get unique API key on your account page

Figura 12.2.4: Registro para la obtención de la clave API

Rellena tus datos como muestra la figura 12.2.5. La información más relevante es la dirección de correo electrónico.

User settings Invoice info

Username alanbritto

Email alan.brito.delgado.1972@gmail.com

Full name Alan Brito

Save

Password

Confirm Password

Figura 12.2.5: Datos de registro

En la pestaña API keys (figura 12.2.6) se encuentra tu clave única por defecto. Guarda esta clave a buen recaudo porque la necesitarás en todas las consultas al servicio web de OpenWeatherMap.

Setup API keys My Services My Payments Billing plans Map editor Block logs Logout

NEW! You can generate as much API keys as needed for your subscription. We accumulate the total loading from all of them.

Create key

Key d3154428888b834152014d4e1646624

Name Default

* Name

Generate

Figura 12.2.6: Clave API

Una vez registrados y obtenida la clave, podemos obtener los datos meteorológicos actuales de casi cualquier ciudad. Veamos el código en PHP.

```

<html>
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>
  <h2>El tiempo en Málaga</h2>
  <?php
    $datos = file_get_contents("http://api.openweathermap.org/data/2.5/weather?q=Sevilla,
Spain&units=metric&appid=<clave API>");
    echo "<h3>Datos en bruto (en formato JSON): </h3>$datos<hr>";
    $tiempo = json_decode($datos);
    echo "<h3>Datos en un objeto: </h3>";
    print_r($tiempo);
    echo "<hr>";
    echo "<h3>Datos sueltos: </h3>";
    echo "Temperatura: ".$tiempo->main->temp."&deg;C<br>";
    echo "Humedad: ".$tiempo->main->humidity."&percnt;<br>";
    echo "Presión: ".$tiempo->main->pressure."&mb<br>";
  ?>
</body>
</html>

```

Este programa genera la salida que se muestra en la [figura 12.2.7](#).

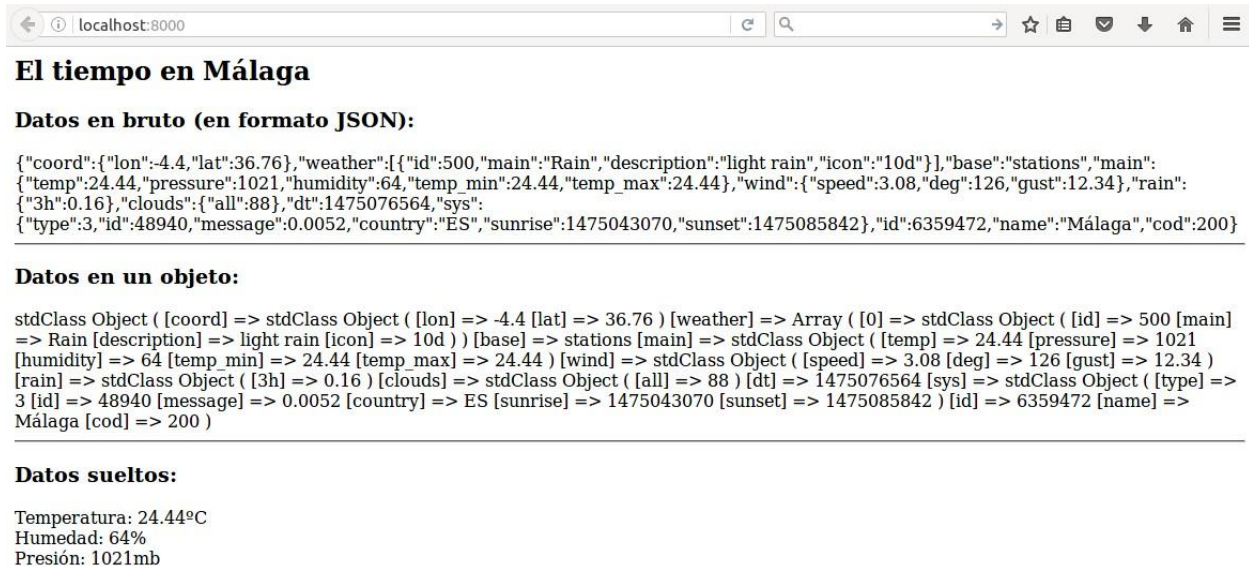


Figura 12.2.7: Datos meteorológicos obtenidos mediante la API de OpenWeatherMap.

Analicemos el programa paso a paso.

La siguiente línea:

```
$datos = file_get_contents("http://api.openweathermap.org/data/2.5/weather?q=Sevilla,  
Spain&units=metric&appid=<clave API>");
```

hace una llamada a la API pasando como parámetros la ciudad (Sevilla), el país (España), tipo de unidad de medida (sistema métrico decimal) y la clave API. En la variable `$datos` se almacena el resultado de la consulta.

Es muy importante sustituir `<clave API>` por el número correspondiente; de lo contrario el programa no funciona. Mediante la línea:

```
$tiempo = json_decode($datos);
```

se decodifican los datos en bruto que están en formato JSON y se guardan en un objeto llamado `$tiempo`.

Para acceder a un dato concreto hay que indicar el atributo dentro del objeto. Por ejemplo, para saber la temperatura, es necesario acceder al atributo `temp` que está dentro de `main` que, a su vez, está dentro del objeto `$temperatura`, tal como se muestra en la siguiente línea.

```
echo "Temperatura: ".$tiempo->main->temp."°C<br>";
```

Es fácil ¿verdad? Prueba ahora a mostrar los datos meteorológicos de tu ciudad.

12.3 Estructura de JSON

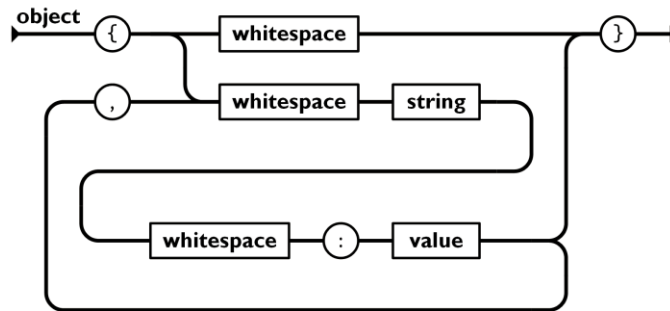
JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

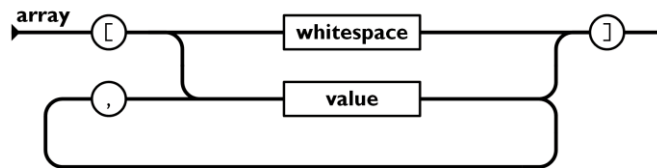
- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

En JSON, se presentan de estas formas:

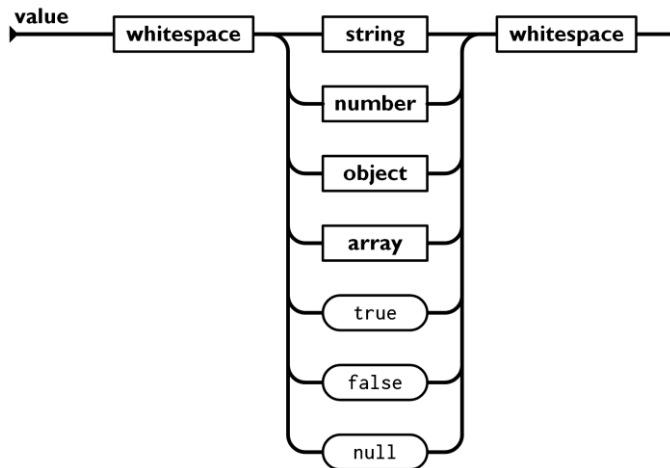
1. Un *objeto(object)* es un conjunto desordenado de pares nombre/valor. Un objeto comienza con {llave de apertura y termine con }llave de cierre. Cada nombre es seguido por :dos puntos y los pares nombre/valor están separados por ,coma.



2. Un *arreglo(array)* es una colección de valores. Un arreglo comienza con [corchete izquierdo y termina con] corchete derecho. Los valores se separan por ,coma.



Un *valor(value)* puede ser una *cadena de caracteres* con comillas dobles, o un *número*, o **true**, o **false**, o **null**, o un *objeto*, o un *arreglo*. Estas estructuras pueden anidarse.



12.4 ¿Qué son las cabeceras o headers HTTP?

Al cargar una página web se producen decenas de peticiones y respuestas HTTP cada una de las cuales lleva cabeceras (headers). Las cabeceras llevan información necesaria para la comunicación y pueden incluir diferentes aspectos como: tipo de navegador que realiza la petición, dirección de la página solicitada, juego de caracteres utilizado, etc.

Cabecera de petición al servidor: La petición que recibe el servidor consta de una línea de petición que contiene alguna información básica sobre la petición y de diversas líneas que constituyen los headers. Un ejemplo puede ser:

```
GET /index.php?option=com_content&view=article&id=57&Itemid=86 HTTP/1.1
Host: aprenderaprogramar.com
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://paginadestino.com/
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120; ciudad=Utrera; usuario=Fernando
Connection: keep-alive
```

La primera línea es la request line, que está formada por 3 partes: el método (GET), la ruta (/index.php?option=com_content&view=article&id=57&Itemid=86) y el protocolo (HTTP/1.1), y el resto de líneas son cabeceras HTTP.

El resto de líneas de cabecera http son pares “Nombre: valor”, que contienen información diversa sobre la petición HTTP y sobre el navegador, y la mayoría son opcionales.

Los datos de la cabecera de petición se pueden consultar en el servidor con el array superglobal \$_SERVER, que es un array asociativo, con distintas entradas con información de la cabecera e información adicional.

Los datos de la cabecera de respuesta se pueden consultar en el cliente que hizo la petición con la variable global \$http_response_header, que es un array no asociativo, del cual la posición que más nos interesa es la primera, ya que muestra el código y mensaje de estado. Consultado el código de respuesta podremos controlar si todo ha ido bien o ha habido algún error:

```
if ($http_response_header[0]=="HTTP/1.1 200 OK") {correcto} else {mostrar error};
```

Cuando la cabecera contiene un código de respuesta de algún error, el navegador nos mostrará un warning que queda muy mal en el contenido de la página, así que para evitarlo, colocaremos una '@' justo delante de la función file_get_contents()

```
$datos = @file_get_contents(http://servicioweb.com);
```

Cabecera de respuesta del servidor: Cada petición que recibe el servidor genera una cabecera de respuesta, que consta de una línea de status con alguna información básica sobre la respuesta del servidor (protocolo usado y código de status), de diversas líneas que constituyen los headers y de una línea en blanco seguida de la respuesta del servidor, normalmente un código HTML extenso. Un ejemplo puede ser:

```
HTTP/1.1 200 OK
Date: Tue, 10 Nov 2015 11:56:29 GMT
Server: Apache/2
X-Powered-By: PHP/5.5.17
P3P: CP="NOI ADM DEV PSAi COM NAV OUR OTRo STP IND DEM"
Expires: Mon, 1 Jan 2001 00:00:00 GMT
```

```
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Last-Modified: Tue, 10 Nov 2015 11:56:30 GMT
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 9237
Keep-Alive: timeout=1, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html lang="es">
<head>... continúa el código html extenso de la página devuelta
```

Al igual que en las peticiones, muchos de los headers son opcionales de modo que la página podría mostrarse en el navegador incluso si no se recibieran.

Definir datos headers de una petición: la función ‘stream_context_create’ crea un contexto para la cabecera de una petición a un servidor, especificando algunos parámetros como el método (GET, POST, PUT o DELETE) o la clave/token a incluir en la cabecera de la petición al servidor, la función debe recibir como parámetro un array asociativo con los pares clave/valor del contexto de la cabecera que se va a crear. Ejemplo:

```
$array_datos=[ 'http'=>[ 'method'=>'GET', 'header'=>'X-Auth-Token: fd7fbd71c601469e3' ] ];
$variable=stream_context_create($array_datos);
$respuesta = file_get_contents("url_del_servidor?param=valor&...", false, $variable);
```

En el ejemplo anterior se crea en \$variable, con la función stream_context_create, el contexto con los datos definidos en \$array_datos (método GET y el token de acceso requerido) y por último se hace la petición al servidor con file_get_contents, usando el contexto definido en \$variable. El segundo parámetro de la función file_get_contents (false), indica que no se especifica una ruta concreta donde buscar en el servidor destino. Si no se especifica el método en la cabecera, por defecto se usará GET.

Software Postman: Es un software gratuito, que permite realizar peticiones de servicios y mostrar los resultados de la respuesta obtenida, configurando cómodamente los parámetros de la url y de la cabecera de manera gráfica.

Para peticiones GET y POST, que son las más comunes, los parámetros que definen esa consulta se especifican en la pestaña ‘Params’ y el servidor podrá acceder a esos parámetros con el array \$_REQUEST, como con cualquier dato enviado desde un formulario.

Las peticiones PUT y DELETE, deben enviar los datos en la pestaña ‘body’ con la opción ‘x-www-form-urlencoded’ para poder acceder en el servicio a los parámetros recibidos, parseando ‘php://input’, si pasamos los parámetros en la pestaña ‘Params’, podemos acceder con \$_REQUEST como hacemos con los protocolos GET y POST, pero solo cuando la petición en el servidor es recibida desde Postman.

12.5 Ejercicios para resolver

Ejercicio 1

Utilizando la API de OpenWeatherMap, realiza una aplicación que muestre la información meteorológica (investiga los distintos parámetros que puede informar la API) de una determinada ciudad. Para la selección de la ciudad se puede utilizar la imagen de un mapa, una o varias listas desplegables, un campo de texto o combinaciones

de los métodos anteriores, de manera que tengamos una aplicación interactiva en la que podamos seleccionar la ciudad y la información meteorológica que queramos conocer de ella.

Ejercicio 2

Realiza una aplicación que haga uso de cualquier API Rest gratuita que encontréis y que proporcione alguna información, debéis procesar esa información y mostrarla formateada correctamente. Si tenéis dificultad para encontrar un servicio gratuito, podéis usar el de la página football-data.org, que ofrece información de fútbol completísima, de manera gratuita, de todas las ligas del mundo, resultados, equipos, jugadores, etc..., además la página contiene mucha información de la API y ejemplos de uso.

12.6 Creación de servicios web

Un programador debe saber tanto consumir servicios web como crear sus propios servicios para que sean consumidos por terceros.

Ilustraremos cómo se pueden proveer servicios mediante un ejemplo.

Supongamos que una empresa dispone de los datos de sus empleados en un archivo en formato CSV. A continuación se muestra un extracto del fichero:

```
"Noel, Julie G.",Proin.mi@dolor.co.uk,659-285422,2805 Ac Avenue,Santo Domingo,Greece,1328
"Cooper, Abraham P.",diam.at@Inmi.com,639-392747,4050 Sit Street,Houtave,Thailand,668
"Curry, Tana A.",at@necleoMorbi.net,689-337896,"689-888 Quis. Rd.",Dampremy,Latvia,2165
"Mathews, Victoria",lorem.sit.amet@amet.com,614-190267,37 Vitae St.,Codognè,Tuvalu,3898
"Peters, Hillary",nonrcu@non.org,630-869705,930-5406 Libero. St.,Bolzano,Greenland,998
"Spencer, Shaine G.",euismod@Vivamus.ca,660-961777,74 Duis Rd.,Burlington,Kazakhstan,2216
"Clark, Morgan R.",non@ipsum.org,628-244525,Ap #888-4243 Pharetra Avenue,Chiari,Guinea,3943
"Anthony, Quail H.",luctus@dolorel.ca,616-967674,Ap #489-4964 Nunc Avenue,Kent,Gabon,2935
"Hurst, Jael ",Duis.a.mi@enean.com,619-955251,5207 Non Rd.,Kamoke,Belarus,1706
```

Cada línea del fichero contiene los datos de un empleado separados por coma. Sobre cada empleado se sabe su nombre completo, dirección de correo electrónico, teléfono, dirección postal, ciudad, país y sueldo neto mensual en euros.

Un departamento de la empresa desea crear un servicio web para que la aplicación de otro departamento distinto pueda consumir información sobre los empleados.

El programa realizado para este cometido es `listadoEmpleados.php`. Este programa devuelve un JSON con los datos de los empleados cuyos nombres contienen la cadena que se pasa en la URL. Por ejemplo, `listaEmpleados.php?nombre=me` devuelve un JSON con todos los datos de los empleados cuyos nombres contienen la cadena “me” (sin distinguir mayúsculas de minúsculas). Se mostrarían los datos de “Jimenez, Noelle J.”, “Cooley, Cameron A.”, “Meyer, Mikayla L.”, etc. Si no se pasa ninguna cadena en la URL, se devuelven los datos de todos los empleados.

El código fuente de `listadoEmpleados.php` se muestra a continuación:

```
<?php
/**
 * Devuelve un JSON con los datos de los empleados cuyos nombres contienen la cadena
 * que* se pasa en la URL.
 *
 * Ejemplo 1:
 *
 * ListaEmpleados.php?nombre=me
 *
 * Devuelve un JSON con todos los datos de los empleados cuyos nombres contienen la
 * cadena "me" (sin distinguir mayúsculas de minúsculas. Se mostrarían los datos de *
 * "Jimenez, Noelle J.", "Cooley, Cameron A.", "Meyer, Mikayla L.", etc.
```

```

*
*   Ejemplo 2:
*
*   ListaEmpleados.php
*
*   Si no se pasa ninguna cadena en la URL, se devuelven los datos de todos los emplea\
dos.
*/
// Abre el fichero con los datos en modo lectura
$f = fopen("datos_empleados.csv", "r");
while (($datosEmpleado = fgetcsv($f, 1000, ","))) {
    // extrae los datos de un empleado
    // En $datosEmpleado[0] está el nombre
    // En $datosEmpleado[1] está el email
    // En $datosEmpleado[2] está el teléfono
    // En $datosEmpleado[3] está la dirección
    // En $datosEmpleado[4] está la ciudad
    // En $datosEmpleado[5] está el país
    // En $datosEmpleado[6] está el sueldo mensual en euros
    // Comprueba si el nombre del empleado contiene la cadena buscada // o si no se ha
    pasado ningún nombre en la URL.
    if ((mb_stripos($datosEmpleado[0], $_GET['nombre']) !== FALSE) ||
!isset($_GET['nombre'])) {
        // Añade los datos del empleado al array
        $empleados[] = [
            'nombre' => $datosEmpleado[0],
            'email' => $datosEmpleado[1],
            'telefono' => $datosEmpleado[2],
            'direccion' => $datosEmpleado[3],
            'ciudad' => $datosEmpleado[4],
            'pais' => $datosEmpleado[5],
            'sueldo' => $datosEmpleado[6]
        ];
    }
} // while
fclose($f);
// Transforma el array con el resultado a formato JSON
echo json_encode($empleados);

```

En la [figura 12.3.1](#) se puede ver el servicio funcionando, observa que la URL es `listadoEmpleados.php?nombre=ca`.

Se muestran todos los datos de los empleados - en formato JSON - cuyos nombres contienen la cadena “ca”.

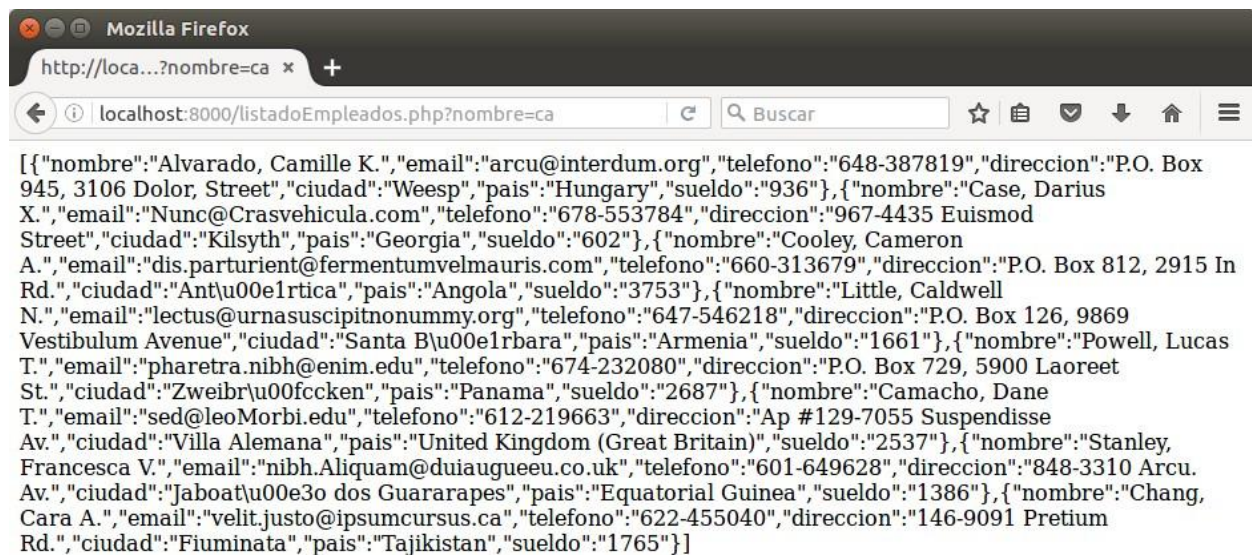


Figura 12.3.1: Datos de empleados en formato JSON.

Cuando creamos los resultados que el servicio tiene que devolver como respuesta en formato JSON, usamos la función `'json_encode($datos)'`, donde `$datos` debe contener una estructura que permita almacenar los datos generados.

El parámetro que pasamos a la función `json_encode`, debe ser un array. Si este array es no-asociativo, al decodificar con `json_decode` obtenemos el array original, pero si el array es asociativo lo que obtenemos al decodificar es un objeto, con los atributos correspondientes a cada elemento del array, por ejemplo:

```
$persona=[ 'nombre' => 'pepe', 'edad'=22];
$codificado=json_encode($persona);
$persona=json_decode($codificado);
echo $persona->nombre . ", " . $persona->edad;
```

Si un array asociativo contiene a su vez otros arrays asociativos en sus valores, se generará una estructura jerárquica de objetos, por ejemplo:

```
$persona=[ 'nombre' => 'pepe', 'edad'=22, 'coche' => [ 'marca' => 'opel', 'motor'=1500]];
$codificado=json_encode($persona);
$persona=json_decode($codificado);
echo $persona->nombre . ", tiene un coche de marca " . $persona->coche->marca;
```

Hay que tener en cuenta que si el array es no-asociativo, se recupera como un array en vez de como un objeto, por lo que podemos crear una estructura de datos devueltos, combinando arrays no-asociativos con arrays asociativos. Lo ideal es que el cliente analice los datos recibidos para ver la estructura de los mismos.

```
$personas[]=[ 'nombre' => 'pepe', 'edad'=22, 'coche' => [ 'marca' => 'opel', 'motor'=1500]];
$personas[]=[ 'nombre' => 'ana', 'edad'=19, 'coche' => [ 'marca' => 'fiat', 'motor'=1800]];
$personas[]=[ 'nombre' => 'juana', 'edad'=27, 'coche' => [ 'marca' => 'mercedes', 'motor'=2500]];
$codificado=json_encode($personas);
$personas=json_decode($codificado);
foreach ($personas as $persona){
    echo $persona->nombre . ", tiene un coche de marca " . $persona->coche->marca;
}
```

Una vez generados los datos y condicionados en formato JSON, para mandar el resultado de vuelta al cliente, el servicio tan solo tiene que imprimir el resultado con 'echo', pero al no estar cargándose este servicio en un navegador, lo que realmente se hace es devolver lo que se está imprimiendo al cliente que ha hecho la petición.

Cuando decodificamos los resultados obtenidos con `json_decode()`, podemos indicar opcionalmente un segundo parámetro booleano, que por defecto estará a `false`, lo que implica que al decodificar obtenemos un objeto con los atributos del array asociativo original, tal como se ha explicado en párrafo anterior, pero si se indica el parámetro como `true` (`'json_decode($datos, true);'`), lo que obtendremos por cada elemento del array principal, es un array asociativo en vez de un objeto. Lo normal es dejarlo por defecto y obtener un objeto, ya que es la forma más cómoda de tratar los datos.

IMPORTANTE: En nuestro entorno de desarrollo, en el cual tenemos ubicado tanto el cliente, como el servidor en la misma máquina física, a la hora de hacer la petición con `file_get_contents()`, tenemos que tener en cuenta que el servidor es una máquina externa, por lo que la dirección especificada en la URL, debe comenzar por 'localhost' seguido de la ruta del servicio php dentro de localhost. Ejemplo 'localhost/ejerciciosPHP/tema13/ejer1Servidor.php'

API RESTFUL completa (GET, POST, PUT, DELETE)

La mayoría de servicios que ofrecen los servidores son para aportar información a los clientes que la solicitan usando el método GET por defecto en la petición, pero a veces se ofrecen servicios que necesitan manipular información en la BD de dicho servidor, para lo cual se usan los métodos POST para añadir información, PUT para modificar o actualizar o DELETE para eliminar información. Para enviar información desde un formulario HTML solo es posible usar los métodos GET o POST, por lo que para usar los métodos PUT o DELETE, es necesario hacerlo con otras tecnologías como AJAX, o bien desde algún software como Postman o desde PHP parseando la información recibida en la cabecera HTTP, como veremos a continuación.

Ejemplo de uso de todos los métodos en servicio completo para consultar productos por nombre o precio, insertar productos, borrar productos y modificar productos añadiendo stock.

index.php (muestra los formularios para realizar las distintas peticiones)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="estilos.css" type="text/css">
  <title>Document</title>
</head>
<body>
  <h1>Consulta de la API productos</h1>
  <div class="contenedor">

    <form action="peticion.php" method="post">
      <h2>Filtro por nombre</h2>
      Nombre <input type="text" name="nombre">
      <input type="submit" name="filtraNombre" value="Filtrar">
    </form><hr>
    <form action="peticion.php" method="post">
      <h2>Filtro por precio:</h2>
```

```

        Mínimo <input type="number" name="min" required>
        Máximo <input type="number" name="max" required>
        <input type="submit" name="filtraPrecio" value="Filtrar">
    </form><hr>
    <form action="peticion.php" method="post">
        <h2>Insertar nuevo producto:</h2>
        Nombre <input type="text" name="nombre" required>
        Precio <input type="number" name="precio" required>
        Stock <input type="number" name="stock" required> <br><br>
        <input type="submit" name="insertar" value="Insertar">
    </form><hr>
    <form action="peticion.php" method="POST">
        <h2>Borrar producto:</h2>
        Nombre <input type="text" name="nombre" required>
        <input type="submit" name="borrar" value="Borrar">
    </form><hr>
    <form action="peticion.php" method="POST">
        <h2>Añadir stock:</h2>
        Nombre <input type="text" name="nombre" required>
        Añadir <input type="number" name="cantidad" require> Unidades
        <input type="submit" name="stock" value="Añadir">
    </form><hr>
</div>
</body>

```

peticion.php (realiza las peticiones al servicio y muestra los resultados)

```

<?php
$url="http://localhost/RESTFULL/consultaProductos.php";
if (isset($_REQUEST['filtraPrecio'])) {
    $parametros = '?min=' . $_REQUEST['min'] . '&max=' . $_REQUEST['max'];
    mostrarDatos($url, $parametros);
} else if (isset($_REQUEST['filtraNombre'])) {
    $parametros = "?nombre=" . $_REQUEST['nombre'];
    mostrarDatos($url, $parametros);
} else if (isset($_REQUEST['insertar'])) {
    $datos = ["nombre"=>$_REQUEST['nombre'], "precio"=>$_REQUEST['precio'], "stock"=>$_REQUEST['stock']];
    pideServicio($url, $datos, "POST");
} else if (isset($_REQUEST['borrar'])) {
    $datos = ["nombre" => $_REQUEST['nombre']];
    pideServicio($url, $datos, "DELETE");
} else if (isset($_REQUEST['stock'])) {
    $datos = ["nombre" => $_REQUEST['nombre'], "cantidad" => $_REQUEST['cantidad']];
    pideServicio($url, $datos, "PUT");
}

function mostrarEstado($codEstado, $mensaje) {
    echo "STATUS: ".$codEstado;
    echo "<br>".$mensaje;
}

function mostrarDatos($url, $parametros) {
    $data = @file_get_contents($url . $parametros);
    $respuesta = json_decode($data);
    $codEstado=substr($http_response_header[0],9,3);
    $mensaje=substr($http_response_header[0],13);
    if ($codEstado==200) {
        echo "<table border='1'><tr><th>Nombre</th><th>Precio</th><th>stock</th></tr>";
        foreach ($respuesta as $producto) {
            echo "<tr><td>".$producto->nombre."</td>";
            echo "<td>".$producto->precio."</td>";
            echo "<td>".$producto->stock."</td></tr>";
        }
        echo "</table>";
    } else {

```

```

        mostrarEstado($codEstado, $mensaje);
    }
    echo "<a href='index.php'><h3>VOLVER A LA PÁGINA DE CONSULTAS</h3></a>";
}

function pideServicio ($url, $datos, $metodo){
    $opciones = [
        "http" => [
            "header" => "Content-type: application/x-www-form-urlencoded\r\n",
            "method" => $metodo,
            "content" => http_build_query($datos)
        ],
    ];
    $contexto = stream_context_create($opciones);
    @file_get_contents($url, false, $contexto);
    $codEstado=substr($http_response_header[0],9,3);
    $mensaje=substr($http_response_header[0],13);
    mostrarEstado($codEstado, $mensaje);
}

```

Dado que la petición con ‘file_get_contents’ usa por defecto el método GET, para enviar los datos en la petición del servicio con un método que sea POST, PUT o DELETE, se crea un array asociativo con dichos datos (\$datos), y la función ‘http_build_query’ convierte este array con parámetros de la petición, en una cadena con formato url (“nombre=\$_REQUEST['nombre']&precio=\$_REQUEST['precio']&...”). Por último se definen las opciones (header, method y content) en un array asociativo (\$opciones) y se crea el contexto con la función ‘stream_context_create’, que se utiliza como parámetro en la petición con ‘file_get_contents’.

Para conocer el resultado de la operación, se consulta el código y mensaje que ha devuelto el servidor en la primera línea de la cabecera de respuesta, que se encuentra en la posición 0 del array ‘\$http_response_header[0]’, extrayendo los caracteres correspondientes (9 a 11 para el código y del 13 al final para el mensaje).

consultaProductos.php (servicio ubicado en el servidor que recibe las peticiones)

```

<?php
require_once 'Producto.php';
$metodo = $_SERVER['REQUEST_METHOD'];
$codEstado=200;
$mensaje = "OK";
if ($metodo == 'GET') {
    if (isset($_REQUEST['min']) && isset($_REQUEST['max'])) {
        $productos = Producto::getProductosFiltroPrecio($_REQUEST['min'], $_REQUEST['max']);
    } else if (isset($_REQUEST['nombre'])) {
        $productos = Producto::getProductosFiltroNombre($_REQUEST['nombre']);
    }
    if (count($productos) == 0) {
        $mensaje = "PRODUCTO NO ENCONTRADO";
        $codEstado = 404;
    } else {
        foreach ($productos as $producto) {
            $devolver[] = ['nombre'=>$producto->getNombre(), 'precio'=>$producto->getPrecio(),
                'stock'=>$producto->getStock()];
        }
    }
} else if ($metodo == 'POST') {
    $producto = Producto::getProductoByNombre($_REQUEST['nombre']);
    if ($producto) {
        $mensaje = "CONFLICTO, PRODUCTO CON MISMO NOMBRE";
        $codEstado = 409;
    } else {
        $producto = new Producto(null, $_REQUEST['nombre'], $_REQUEST['precio'], null, $_REQUEST['stock']);
        $producto->insert();
        $mensaje = "PRODUCTO INSERTADO CORRECTAMENTE";
    }
}

```

```

        $codEstado = 200;
    }
} else if ($metodo == 'DELETE') {
    //Para los métodos GET y POST existe $_REQUEST, pero para PUT y DELETE no, así que parseamos 'php://input'
    parse_str(file_get_contents("php://input"), $parametros);
    $producto = Producto::getProductoByNombre($parametros['nombre']);
    if ($producto) {
        $producto->delete();
        $mensaje = "PRODUCTO BORRADO CORRECTAMENTE";
        $codEstado=200;
    } else {
        $mensaje = "PRODUCTO NO ENCONTRADO";
        $codEstado=404;
    }
}
} else if ($metodo == 'PUT') {
    //Para los métodos GET y POST existe $_REQUEST, pero para PUT y DELETE no, así que parseamos 'php://input'
    parse_str(file_get_contents("php://input"), $parametros);
    $producto = Producto::getProductoByNombre($parametros['nombre']);
    if ($producto) {
        $producto->añade($parametros['cantidad']);
        $mensaje = "STOCK AÑADIDO CORRECTAMENTE";
        $codEstado=200;
    } else {
        $mensaje = "PRODUCTO NO ENCONTRADO";
        $codEstado=404;
    }
}

setCabecera($codEstado, $mensaje);
echo json_encode($devolver);

function setCabecera($codEstado, $mensaje) {
    header("HTTP/1.1 $codEstado $mensaje");
    header("Content-Type: application/json; charset=utf-8");
}

```

En el servicio lo primero que hacemos es analizar el método de la petición recibida consultando el valor de `$_SERVER['REQUEST_METHOD']`, y para acceder a los datos parametrizados definidos en el contexto creado en la cabecera de la petición desde el cliente, usamos `parse_str(file_get_contents("php://input"), $parametros)`; generando el array asociativo `$parametros`, que contiene los parámetros recibidos.

Para el tratamiento de errores, devolvemos en la cabecera de respuesta el código y mensaje asociado, resultantes de la operación realizada, que almacenamos previamente en las variables `'mensaje'` y `'codEstado'`. En el array que mandamos codificado en json se almacena el array con los productos filtrados con la siguiente estructura:

```

$respuesta=[//array no asociativo con todos los productos
    [ 'nombre' => 'teclado', 'precio' => 25, 'stock' => 10 ],
    [ 'nombre' => 'raton', 'precio' => 15, 'stock' => 15 ],
    [ 'nombre' => 'monitor', 'precio' => 85, 'stock' => 5 ]
];

```

Algunos de los códigos de estado más usados son:

- 200 Correcto
- 400 Solicitud incorrecta: el cliente envió una solicitud no válida, como la falta de un cuerpo o parámetro de solicitud requerido
- 401 No autorizado: el cliente no pudo autenticarse con el servidor
- 403 Prohibido: el cliente se autenticó pero no tiene permiso para acceder al recurso solicitado
- 404 no encontrado: el recurso solicitado no existe

- 409 Conflicto: la operación genera un conflicto (como insert duplicado)
- 500 Internal Server Error: se produjo un error genérico en el servidor
- 503 Servicio no disponible: el servicio solicitado no está disponible

Aunque no usaremos todos los códigos, está bien saber que existen cinco categorías de códigos de estado:

- 1xx : el servidor reconoce una solicitud
- 2xx (correcto): el servidor completó la solicitud como se esperaba
- 3xx (redirección): el cliente debe realizar más acciones para completar la solicitud
- 4xx (error del cliente): el cliente envió una solicitud no válida
- 5xx (error del servidor): el servidor no pudo cumplir una solicitud válida debido a un error con el servidor

12.7 Ejercicios para resolver

Ejercicio 1

Crea un servicio web para pasar de euros a pesetas y de pesetas a euros. Si se pasa por la URL una cantidad en euros, el programa lo debe convertir en pesetas y viceversa.

Ejercicio 2

Crea un servicio web que proporcione de forma aleatoria un mazo de cartas de la baraja española. Por la URL se pasa el número de cartas que se quiere obtener y la aplicación provee el palo y la figura de cada una de las cartas. Se supone que el mazo se obtiene de una baraja, por tanto, las cartas no se pueden repetir. Si el número que se pasa en la URL es menor que 1 o mayor que 40, se debe devolver un error.

Ejercicio 3

Crea un servicio web que proporcione información sobre los productos de la base de datos, del carrito de la compra. Se devolverá en formato JSON, el nombre, precio y url de la imagen, de cada uno de los productos seleccionados.

La petición podrá ser por nombre o por precio. Por nombre el servicio devolverá los productos cuyo nombre contenga la cadena recibida por parámetro. Y por precio el servicio devolverá los productos cuyo precio este dentro del rango mínimo y máximo recibido por parámetro.

Para poder usar el servicio, el cliente debe haberse registrado previamente y recibido su 'TOKEN' de acceso. Estos tokens están almacenados en una tabla nueva llamada 'clientes' con tres campos, 'nombre', 'token' y 'peticiones'. En cada petición el cliente debe mandar el token y el servidor debe comprobar que es un token válido correspondiente a un cliente registrado, y sumar uno al campo peticiones en la tabla, para llevar un control de las peticiones que realiza cada cliente.

Ejercicio 4

Crea un servicio web que proporcione información sobre las matriculaciones de los alumnos en asignaturas del ejercicio 4 de la unidad anterior y permita realizar algunas operaciones en los datos.

Se podrán realizar peticiones sobre las siguientes consultas:

- Los alumnos de un grupo pasado por parámetro
- Alumnos cuyo nombre contenga la cadena pasada por parámetro
- Asignaturas de un alumno pasado por parámetro.

El servicio también permitirá realizar las siguientes acciones:

- Matriculación de un alumno en una asignatura.
- Cambio de grupo de un alumno
- Borrar un alumno

Devolver el código de estado de la petición y el mensaje asociado a dicho código, junto con los datos. Probar y guardar todas las peticiones posibles en la aplicación Postman.

Como segunda parte del ejercicio, realiza una página que actúe como cliente realizando peticiones al servidor, a través de formularios.