

PHP

Material creado para **Desarrollo Web en Entorno Servidor**

- Desarrollo de Aplicaciones Web II -



Índice

1. Conceptos previos	3
1.1 Internet y la arquitectura Cliente-Servidor	3
1.2 Páginas estáticas y dinámicas	4
1.3 Tecnologías Web	4
1.4 Frameworks	5
2. Introducción al lenguaje PHP	7
2.1 El lenguaje	7
2.2 Utilidades	7
2.3 La comunidad	8
3. Herramientas de programación	9
4. Sintaxis de PHP	10
4.1 Sintaxis básica	10
4.1.1 Delimitadores	10
4.1.2 Comentarios	10
4.1.3 Fin de instrucción	11
4.1.4 Mostrar por pantalla	11
4.2 Variables	12
4.3 Operadores	13
4.3.1 Aritméticos	13
4.3.2 De cadenas	13
4.3.3 Asignación	13
4.3.4 De incremento y decremento	14
4.3.5 De Comparación	14
4.3.6 Lógicos	14
4.3.7 Para arrays	15
4.4 Constantes	15
4.5 Variables predefinidas	16
4.6 Tipos de datos	18
4.6.1 Integer	19
4.6.2 Float	19
4.6.3 Boolean	19
4.6.4 String	20
4.6.5 Array	21
4.6.5.1 Array escalar	22
4.6.5.2 Array asociativo	23
4.6.5.3 Array multidimensional	23
4.6.5.4 list()	24
4.6.6 Object	25
4.7 Funciones para variables	25
5. Estructuras de Control	27
5.1 Estructuras de selección (Sentencias condicionales)	27
5.1.1 If	27

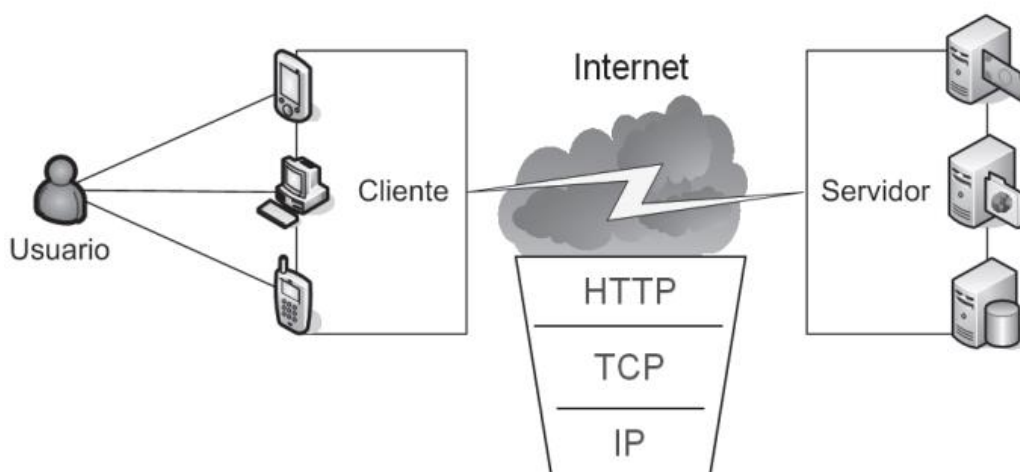
5.1.2 Switch	29
5.2 Estructuras repetitivas o de recorrido (Sentencias de bucle)	30
4.2.1 While	30
5.2.2 Do-While	30
5.2.3 For	31
5.2.4 Foreach	32
5.3 Instrucciones para ruptura, finalización, salto y retorno	34
5.3.1 Break	34
5.3.2 Continue	35
6. Funciones	36
6.1 Devolver valores	36
6.2 Argumentos de funciones	37
6.2.1 Argumentos predeterminados	38
6.3 Ámbito de las variables	38
6.3.1 Variables locales	38
6.3.2 Variables globales	39
6.3.3 Variables estáticas	40
6.4 Recursividad	40
6.5 Uso de librerías de funciones	40
6.6 Funciones para tratamiento de texto y arrays	42
6.6.1 Funciones para el tratamiento de arrays.	42
6.6.2 Funciones para el tratamiento de cadenas de texto	44
6.7 Otras funciones	46
7. Paso de información entre documentos	47
7.1 Paso de información con formularios: GET y POST	47
7.2 Paso de información con COOKIES	49
7.3 Paso de información con SESIONES	54
8. Tratamiento de ficheros	58
8.1 Funciones para lectura y escritura de ficheros	58
8.1.1 Apertura de un fichero	58
8.1.2 Cierre de un fichero	59
8.1.3 Lectura de un fichero	60
8.1.4 Recorrer un fichero	62
8.1.5 Escritura en un fichero	63
8.1.6 Borrado de un fichero	63
9. Programación orientada a objetos con PHP	64
9.1 Clases, Métodos y propiedades	64
9.1.1 Instancia de clase	64
9.1.2 Método constructor	65
9.1.3 Método destructor	65
9.2 Herencia	66
10. PHP y las Bases de Datos	67
Bibliografía	68

1. Conceptos previos

1.1 Internet y la arquitectura Cliente-Servidor

La World Wide Web está formada por un conjunto de recursos interconectados.

Se utiliza el modelo Cliente/Servidor, en el que el cliente consume servicios que proporciona el servidor o servidores. Esto se realiza a través del intercambio de mensajes y la utilización de protocolos tales como HTTP, TCP, IP, ...



Normalmente el cliente inicia el intercambio de información, solicitando datos al servidor, que responde enviándole uno o más flujos de datos.

Cuando el navegador solicita a un servidor web una página, es posible que antes de enviarla haya tenido que ejecutar algún programa para obtenerla. Ese programa es el que genera, en parte o en su totalidad, la página web que llega a tu equipo. En estos casos, el código se ejecuta en el entorno del servidor web.

Además, cuando una página web llega al navegador, es también posible que incluya algún programa o fragmentos de código que se deban ejecutar. Ese código, normalmente en lenguaje JavaScript, se ejecutará en el navegador y, además de poder modificar el contenido de la página, también puede llevar a cabo acciones como la animación de textos u objetos de la página o la comprobación de los datos que introduces en un formulario.

Por ejemplo, en el correo web, el programa que se encarga de obtener tus mensajes y su contenido de una base de datos se ejecuta en el entorno del servidor, mientras

que tu navegador ejecuta, por ejemplo, el código encargado de avisarte cuando quieres enviar un mensaje y te has olvidado de poner un texto en el asunto.

1.2 Páginas estáticas y dinámicas

Una aplicación Web consiste en un conjunto de páginas web que pueden ser estáticas y dinámicas o únicamente dinámicas.

Una página web estática es aquella que no se ve modificada cuando el usuario la solicita, es decir, el servidor la envía al navegador web solicitante sin realizarle ninguna modificación. Estas páginas están almacenadas en su forma definitiva, tal y como se crearon, y su contenido no varía. Muestran una misma información cada vez que se carguen, únicamente pueden cambiar si un programador la modifica y actualiza su contenido.

En cambio, una página web dinámica se ve modificada en el servidor antes de enviarse al navegador solicitante. Su contenido cambia en función de diversas variables, como por ejemplo el navegador que estamos usando, el usuario que se ha identificado, etc.

Dentro de las páginas web dinámicas tenemos dos tipos:

- Las que incluyen código que se ejecuta en el navegador. Dentro del HTML (o XHTML) se incluye código ejecutable (en JavaScript) y se descarga junto con la página. El navegador, al mostrar la página en pantalla, ejecuta el código.
- En las que el HTML se forma como resultado de la ejecución de un código, ejecución que tiene lugar en el servidor web.

1.3 Tecnologías Web

Las tecnologías en las que el servidor procesa alguna información (desde el formateo hasta la gestión del contenido, consultas a la base de datos, etc.) son las llamadas Tecnologías del lado del servidor. Algunos ejemplos de estas tecnologías son:

- PHP: Lenguaje de programación del lado del servidor. Es fácil de usar y tiene muchos frameworks disponibles, como Laravel y Symfony.

- Python: Lenguaje de programación que se utiliza mucho en el desarrollo web. Ofrece una sintaxis clara y legible. Tiene frameworks, como Django y Flask.
- Ruby: Lenguaje de programación conocido por su elegancia y legibilidad. Ruby on Rails es un framework muy popular.
- Java: Java es un lenguaje de programación orientado a objetos que se utiliza ampliamente en el desarrollo de aplicaciones empresariales. Tiene el framework Spring.
- Node.js: Es una plataforma que permite ejecutar JavaScript en el lado del servidor. Es muy popular para el desarrollo de aplicaciones web en tiempo real y aplicaciones de una sola página (Single-Page Applications).
- ASP.NET: Framework desarrollado por Microsoft para crear aplicaciones web. Utiliza el lenguaje de programación C# y se basa en el patrón de diseño Modelo-Vista-Controlador (MVC). Se utiliza en el desarrollo de aplicaciones empresariales.

1.4 Frameworks

Para hacer un desarrollo se suelen usar frameworks, que son colecciones de funciones, objetos, reglas y otros componentes que se han diseñado para resolver problemas comunes y acelerar el desarrollo simplificando tareas.

Proporcionan un alto nivel de abstracción para simplificar el desarrollo de aplicaciones. Así los desarrolladores se pueden enfocar en la lógica de negocio y la funcionalidad específica de la aplicación, en lugar de ocuparse de los detalles técnicos más básicos.

Características comunes que suelen ofrecer los frameworks web:

- Enrutamiento: Permiten definir las rutas URL de la aplicación y asociarlas con controladores o acciones específicas.
- Controladores: Proporcionan la lógica de control de la aplicación, procesando las solicitudes y gestionando las respuestas.
- Modelos: Permiten interactuar con la base de datos y realizar operaciones de lectura y escritura.
- Vistas: Se encargan de la presentación de los datos al usuario, generando las respuestas HTML, XML u otros formatos.

- Gestión de sesiones y autenticación: Facilitan la gestión de sesiones de usuario y la autenticación de usuarios.
- Validación de datos: Proporcionan mecanismos para validar los datos recibidos de los formularios o las solicitudes.
- Seguridad: Ofrecen funciones y herramientas para proteger la aplicación contra ataques comunes, como inyección de código SQL.

Nosotros utilizaremos el Framework [Laravel](#) de PHP.

2. Introducción al lenguaje PHP

2.1 El lenguaje

PHP = Hypertext Preprocessor.

Permite diseñar páginas web dinámicas: genera páginas bajo petición capaces de responder de manera inteligente a las demandas del cliente y capaces de automatizar tareas.

PHP es un lenguaje de programación interpretado de guiones de lado del servidor, por lo que requiere tener un servidor web para poder trabajar con él, aunque pretendamos hacerlo en local.

Su sintaxis está basada en la de C / C++ y es similar a la de Java.

Utilizaremos el servidor Apache, que arrancaremos desde XAMPP.

PHP permite embeber fragmentos de código en una página HTML para realizar determinadas acciones de un modo sencillo y eficaz. Lo que hacemos es introducir scripts dentro del código HTML. Además, PHP tiene muchas funciones para el manejo de bases de datos.

2.2 Utilidades

Permite por ejemplo recopilar datos de formularios, generar contenido dinámico para las páginas (interactuación con el usuario), enviar y recibir cookies, etc.

Algunos usos de PHP:

- Desarrollo de sitios web: Creación de sitios web dinámicos e interactivos. Puede generar contenido en función de la entrada del usuario, realizar validaciones, interactuar con bases de datos, etc.
- Muchos sistemas de gestión de contenido (CMS) están escritos en PHP: WordPress, Drupal y Joomla. Los CMS permiten crear y gestionar fácilmente sitios web sin conocimientos de programación.
- Aplicaciones web: Permite desarrollar aplicaciones web completas.
- Acceso a bases de datos: Soporta acceso a muchas bases de datos, nosotros usaremos MariaDB (compatible con MySQL).

Para trabajar con PHP necesitamos:

- Analizador (o intérprete) de PHP
- Servidor web

En **XAMPP** tenemos el servidor apache y el intérprete (o analizador) de PHP ya integrado. Además, incluye el servidor de base de datos MariaDB, que usaremos con posterioridad.

PHP puede emplearse en todos los sistemas operativos principales, incluyendo Linux, muchas variantes de Unix (HP-UX, Solaris, OpenBSD,...), Microsoft Windows, Mac OS X, ... PHP admite la mayoría de servidores web de hoy en día, incluyendo Apache. Además, se tiene la posibilidad de utilizar programación por procedimientos, programación orientada a objetos (POO), o una mezcla de ambas.

2.3 La comunidad

La comunidad de desarrolladores de PHP es muy amplia y además de soporte da acceso a recursos. Para acceder a ella:

- Sitio web oficial de PHP: php.net
- Foros: PHP Freaks (phpfreaks.com) y PHPBuilder (phpbuilder.com)
- Comunidades en línea: Stack Overflow (stackoverflow.com) para obtener información y GitHub (github.com) para explorar proyectos o colaborar con otros desarrolladores.
- Redes sociales y grupos en línea

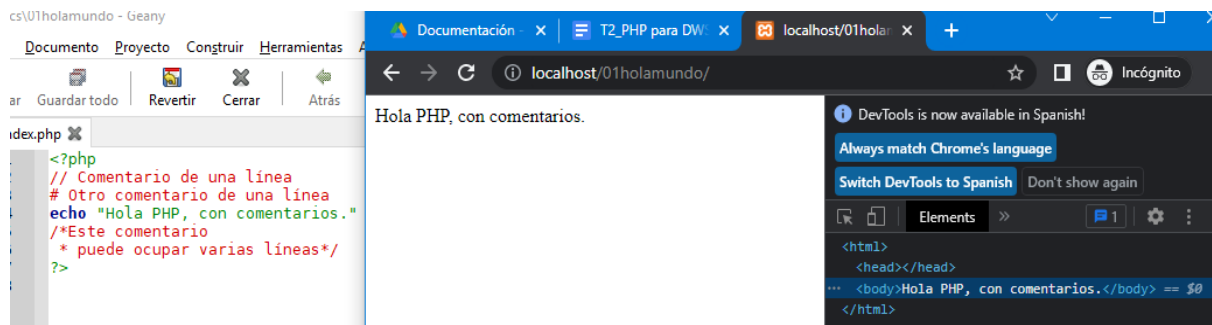
3. Herramientas de programación

Podemos utilizar, entre otras, Geany, NetBeans, Visual Studio Code o Eclipse.

No es imprescindible utilizar un IDE para programar. Podemos usar un simple editor de texto pero el entorno de desarrollo proporciona ciertas ventajas:.

- Resaltado de texto. Muestra con distinto color o tipo de letra los diferentes elementos del lenguaje: sentencias, variables, comentarios, etc. También genera indentado automático para diferenciar de forma clara los distintos bloques de un programa.
- Completado automático. Detecta qué estás escribiendo y cuando es posible te muestra distintas opciones para completar el texto.
- Navegación en el código. Permite buscar de forma sencilla elementos dentro del texto, por ejemplo, definiciones de variables.
- Comprobación de errores al editar. Reconoce la sintaxis del lenguaje y revisa el código en busca de errores mientras lo escribes.

IDE de código abierto muy usados son NetBeans y Eclipse. En el aula se encuentra NetBeans instalado, así como Geany, cualquiera de las opciones es válida.



4.1.3 Fin de instrucción

El fin de sentencia se marca con el carácter punto y coma (;) o aprovechando la etiqueta de cierre ?>.

4.1.4 Mostrar por pantalla

Para mostrar por pantalla se utiliza la instrucción **echo** (o **print**) seguida del contenido entre comillas dobles o simples.

```
echo "Hola PHP"
```

```
echo "Esto ocupa múltiple líneas.
```

```
Los saltos de línea no se mostrarán";
```

La instrucción echo permite concatenar varios parámetros con el operador punto (.):

```
echo "Hola"." Mundo"; // mostrará Hola Mundo
```

Se pueden usar comillas dobles o simples pero si queremos imprimir una de ellas por pantalla usaremos las dos.

```
echo 'Este texto viene "entrecorillado" ';
```

Si la cadena está entre comillas simples, se imprimirá literalmente. Si deseamos que se sustituyan las variables que contenga por sus valores hay que usar comillas dobles.

```
$cant=8;
```

```
echo 'Son $cant euros'; //dará como resultado Son $cant euros.
```

```
echo "Son $cant euros"; //imprimirá Son 8 euros.
```

Ejemplo comillas

4.2 Variables

A las variables les pone delante el signo del dólar (\$). Hay que tener en cuenta que distingue entre mayúsculas y minúsculas, no es lo mismo \$var que \$Var. Y además no pueden comenzar por número.

Recordamos que una variable es una referencia a un espacio en la memoria en el que se encuentra almacenado un dato.

En PHP no hace falta declarar las variables antes de usarlas. La primera vez que se usan ya se declaran automáticamente.

```
$var1 = 123;
$var2 = "Hola";
$var3= $var1 * 2;
$var4 = 12.3;
$var5[1] = "uno";
```

Están débilmente tipadas, al otorgarles un valor se les asigna un tipo. Es decir, el tipo está asociado al contenido, no a la variable en sí. Una variable se puede reutilizar asignándole a lo largo del tiempo datos de distinto tipo.

```
$mi_var = 'Inicializamos como cadena de caracteres';
$mi_var = 3; //pasa a ser entero
$mi_var = 3.14; //aquí es un float
$mi_var = new MiClase(); //ahora es un objeto
```

Hay que tener cuidado con los tipos ya que PHP hace las transformaciones que considera.

```
$mivar = '3';
$mivar = (2 + mivar); //¿cual será el resultado? no da error, el resultado es 5
```

Sin embargo, podemos forzar la conversión a un tipo específico.

```
$mi_var1 = (string) 3;
$mi_var2 = 2;
settype ($mi_var2, "double");
```

Las variables tienen un ámbito local, aunque pueden ser accedidas de forma global en todo nuestro código si usamos la palabra **global** delante.

```
global $var = 7;
```

4.3 Operadores

4.3.1 Aritméticos

Son los de la aritmética básica y se aplican a variables y constantes numéricas:

Operador	Descripción	Ejemplo
----------	-------------	---------

+	Suma	$\$a + B + 5$
-	Resta o negación	$-\$a$ $\$a - B - 5$
*	Multiplicación	$\$a * B * 5$
/	División	$\$a / B$
%	Módulo o mod (resto de la división)	$\$a \% B$

El operador de división (“/”) devuelve un valor flotante en todos los casos, incluso si los dos operandos son enteros.

Ejemplo operadores aritméticos

4.3.2 De cadenas

Se utiliza el punto (.) para unir cadenas

4.3.3 Asignación

El operador de asignación es =

Junto con el operador básico de asignación, existen “operadores combinados” para todas las operaciones de aritmética binaria y de cadenas, que le permiten usar un valor en una expresión y luego definir su valor como el resultado de esa expresión.

Operador	Ejemplo	Equivale a
+=	$\$a += \b	$\$a = \$a + \$b$
-=	$\$a -= \b	$\$a = \$a - \$b$
*=	$\$a *= \b	$\$a = \$a * \$b$
/=	$\$a /= \b	$\$a = \$a / \$b$
%=	$\$a \% = \b	$\$a = \$a \% \$b$
.=	$\$a .= \b	$\$a = \$a . \$b$

4.3.4 De incremento y decremento

Operador	Ejemplo	Efecto
++	$++\$a$ (Pre-incremento)	Incrementa $\$a$ en 1 y luego lo devuelve
++	$\$a++$ (Post-incremento)	Devuelve $\$a$ y luego lo incrementa en 1

-	-\$a (Pre-decremento)	Decrementa \$a en 1 y luego lo devuelve
-	\$a- (Post-decremento)	Devuelve \$a y luego lo decrementa en 1

4.3.5 De Comparación

Permiten comparar dos valores y devuelven true o false.

Si se compara un entero con una cadena, la cadena es convertida a un número.

Operador	Descripción	Ejemplo
==	Igual true si los operandos son iguales, después de la manipulación de tipos	\$a == B
===	Idéntico true si los operandos son iguales y del mismo tipo	\$a === B
!= <>	Diferente true si los operandos son diferentes, después de la manipulación de tipos	\$a != B \$a <> 5
!==	No idéntico true si los operandos son distintos o no son del mismo tipo	\$a !== B
>	Mayor que	\$a > B
>=	Mayor o igual que	\$a >= B
<	Menor que	\$a < B
<=	Menor o igual que	\$a <= B

Ejemplo operadores de comparación

4.3.6 Lógicos

Estos operadores primero convierten a sus operandos en booleanos, luego realizan la comparación y devuelven el resultado, que también es booleano.

Operador	Descripción	Ejemplo
and &&	And (y) true si todos los operandos son true	\$a and B \$a && B
or 	Or (o inclusivo) true si cualquier operando es true	\$a or B \$a B
xor	Xor (o exclusivo)	\$a xor B

	true si sólo uno de los operandos es true	
!	Not (no) true si el operando no es true	! \$a

4.3.7 Para arrays

Operador	Ejemplo	Equivale a
+	\$a + \$b	Unión de \$a y \$b (anexa al final de \$a)
==	\$a == \$b	True si \$a y \$b tienen las mismas parejas llave/valor
===	\$a === \$b	True si \$a y \$b tienen las mismas parejas llave/valor en el mismo orden y mismos tipos
!= <>	\$a != \$b	True si \$a no es igual a \$b
!==	\$a !== \$b	True si \$a no es idéntico a \$b

4.4 Constantes

Una constante es una variable que mantiene el mismo valor durante toda la ejecución. PHP distingue entre mayúsculas y minúsculas pero los nombres de constantes se suelen indicar en mayúsculas.

Suelen usarse con un ámbito global (están disponibles desde cualquier parte del código, ya sea dentro de una función, en una clase o en cualquier otro lugar del script) pero podemos hacerlo en un ámbito local.

Hay dos formas de definir las:

define (“NOMBRE”, valor)

`define (“IVA”, “21%”);`

`echo IVA; //no van entre las comillas como en la definición o las variables`

const NOMBRE = valor

`const VALOR_MINIMO = 0.0;`

`echo VALOR_MINIMO;`

Con esta opción se pueden definir en ámbito local en una clase, no se puede definir así en una función y si se hace en el código general se genera una constante global.

Para saber si una constante está creada: **defined** (“NOMBRE_CONSTANTE”): devuelve verdadero si la constante ya está definida.

Ejemplo constantes

PHP tiene constantes predefinidas, que muestran valores específicos propios de la configuración de nuestro sistema:

- `PHP_VERSION` // Versión del intérprete de PHP de nuestro servidor
`echo “Estamos trabajando con la versión: “ . PHP_VERSION;`
- `PHP_OS` // Sistema operativo de nuestro servidor
`echo “En el sistema operativo: “ . PHP_OS;`
- `PHP_LIBDIR` // Ruta en la que se almacenan las librerías de PHP

PHP también tiene constantes mágicas, que muestran valores dependiendo de dónde se ejecuten. Se acompañan por la izquierda y por la derecha de dos guiones bajos seguidos:

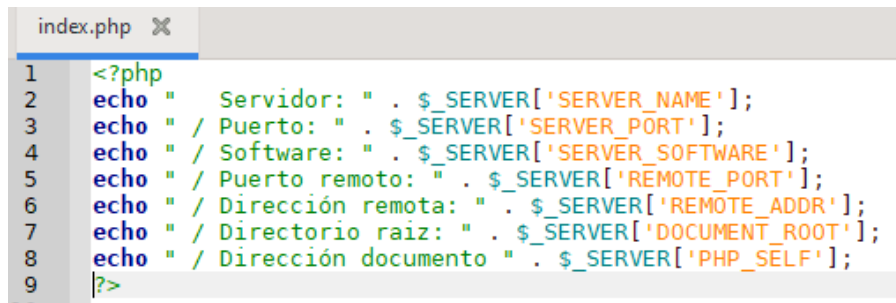
- `__LINE__` // Indica el número de línea actual
`echo “Esta es la línea: “ . __LINE__ . “ del código”;`
- `__FILE__` // Nombre y ruta completa del archivo
- `__CLASS__` // Nombre de la clase en la que nos encontramos

Ejercicio 1

4.5 Variables predefinidas

PHP dispone de variables que ya se encuentran definidas y podemos usar:

- `$_SERVER`: Array asociativo con las variables pasadas desde el servidor
 - `SERVER_NAME`: Nombre del equipo sobre el que se ejecuta
 - `SERVER_PORT`: Puerto usado por el servidor Web
 - `SERVER_SOFTWARE`: Software en uso en el servidor
 - `REMOTE_PORT`: Puerto que utiliza el cliente para comunicarse
 - `REMOTE_ADDR`: Dirección del cliente
 - `DOCUMENT_ROOT`: Directorio raíz del documento en el que se ejecuta el script
 - `PHP_SELF` : El nombre del archivo de script ejecutándose actualmente, junto con su ruta, relativa al directorio raíz de documentos del servidor.



```

1 <?php
2 echo "  Servidor: " . $_SERVER['SERVER_NAME'];
3 echo " / Puerto: " . $_SERVER['SERVER_PORT'];
4 echo " / Software: " . $_SERVER['SERVER_SOFTWARE'];
5 echo " / Puerto remoto: " . $_SERVER['REMOTE_PORT'];
6 echo " / Dirección remota: " . $_SERVER['REMOTE_ADDR'];
7 echo " / Directorio raíz: " . $_SERVER['DOCUMENT_ROOT'];
8 echo " / Dirección documento " . $_SERVER['PHP_SELF'];
9 ?>

```

[Más sobre \\$_SERVER](#)

- **\$GLOBALS:** Array asociativo con todas las variables disponibles en el ámbito global. Con ella se puede acceder a las variables globales desde cualquier parte del script.
- **\$_SESSION:** Array asociativo con las variables de la sesión
- **\$_FILES:** Array asociativo con información acerca de los archivos cargados a una página
- **\$_ENV:** Array asociativo con información acerca de las variables definidas en el entorno bajo el que está siendo ejecutado el intérprete PHP (definidas directamente sobre el sistema)
- **\$_GET:** Array asociativo con las variables pasadas a través del método GET
- **\$_POST:** Array asociativo con las variables pasadas a través del método POST
- **\$_COOKIES:** Array asociativo que con las variables pasadas a través de cookies.
- **\$_REQUEST:** contiene el contenido de **\$_GET**, **\$_POST** y **\$_COOKIE**.

[Ejemplo variables predefinidas](#)

4.6 Tipos de datos

Tipos de datos simples, escalares o de un único valor:

- integer
- float
- string
- boolean

Tipos de datos compuestos:

- array
- object

Tipos de datos especiales:

- resource

Al realizar una operación con variables de distintos tipos, ambas se convierten primero a un tipo común.

```
$mi_entero = 3;
```

```
$mi_real = 2.3;
```

```
$resultado = $mi_entero + $mi_real; // $resultado es de tipo real (valor 5.3)
```

Estas conversiones de tipo también se pueden realizar de forma forzada, escribiendo entre paréntesis el tipo deseado antes de la variable a convertir:

```
$mi_entero = 3;
```

```
$mi_real = 2.3;
```

```
$resultado = $mi_entero + (int) $mi_real; // $resultado es entero (valor 5)
```

Los siguientes casting de tipos están permitidos:

(int), (integer) - forzado a integer

(bool), (boolean) - forzado a boolean

(float), (double), (real) - forzado a float

(string) - forzado a string

(array) - forzado a array

(object) - forzado a object

(unset) - forzado a NULL

Valor de \$var	(int) \$var	(bool) \$var	(string) \$var	(float) \$var
null	0	false	""	0
true	1	true	"1"	1
false	0	false	""	0
0	0	false	"0"	0
3.8	3	true	"3.8"	3.8
"0"	0	false	"0"	0
"10"	10	true	"10"	10
"6 metros"	6	true	"6 metros"	6
"hola"	0	true	"hola"	0

4.6.1 Integer

Número entero (no decimal). Son los enteros, tanto positivos como negativos, incluido el cero. Puede especificarse en formato decimal (base 10), hexadecimal (base 16, se indica con prefijo 0x) u octal (base 8, se indica con prefijo 0) y ser precedido opcionalmente por los símbolos + ó -.

```
$num = 1234;
```

```
$num = -123;
```

```
$num = 0123; // numero octal (equivalente al 83 decimal)
```

```
$num = 0x1A; // numero hexadecimal (equivalente al 26 decimal)
```

Las funciones echo y print muestran por defecto la información en decimal aunque internamente se haya almacenado con otro formato.

4.6.2 Float

Los números en coma flotante se pueden representar en notación normal y científica

```
$a = 1.234; // notación normal
```

```
$b = 1.2e3; // notación científica
```

4.6.3 Boolean

Expresa un valor de verdad. Puede ser TRUE o FALSE. Además, cualquier número entero se considera como true, salvo el 0 que es false.

```
$bool = TRUE; // un valor booleano con valor verdadero
```

4.6.4 String

Representa tanto caracteres simples como cadenas de caracteres sin tamaño definido. No existe el tipo *carácter*.

Para PHP una variable es de tipo string siempre que su valor vaya entrecomillado. Podemos utilizar tanto comillas simples como dobles, lo cual nos permite utilizar unas dentro de las otras sin cerrar el string. Sin embargo PHP interpreta de manera distinta el uso de las comillas dobles y las simples: los strings con comillas dobles sustituyen los nombres de las variables por su contenido, mientras que las comillas simples muestra el contenido de forma literal, es decir, deja el nombre de la variable.

- ❖ Las comillas simples no tienen en cuenta ningún carácter de escape, excepto a la comilla simple y \ que se escapa por una contrabarra.

```
<?php
echo 'La comilla simple escapa solo el carácter de comilla simple \' y el de contrabarra \\. ';
echo "La comilla doble lo hace también \" así y el de contrabarra \\ así.";
?>
```

```
La comilla simple escapa solo el carácter de comilla simple ' y el de contrabarra \.
La comilla doble lo hace también " así y el de contrabarra \ así.
```

- ❖ Las comillas dobles hacen expansión de variables. Cualquier nombre de variable dentro de la cadena se sustituye por su valor (siempre que la variable sea de un tipo simple, no compuesto).

```
<?php
$nombre = 'Pepe';
echo "Hola $nombre, está con comillas DOBLES y SUSTITUYE la variable";
echo ' pero ahora $nombre, está con comillas SIMPLES y NO SUSTITUYE la variable';
?>
```

```
Hola Pepe, está con comillas DOBLES y SUSTITUYE la variable pero ahora
$nombre, está con comillas SIMPLES y NO SUSTITUYE la variable
```

- ❖ Dentro de unas comillas no es necesario escapar las otras.

```
<?php
echo ' Pongo comilla doble " dentro de simple. ';
echo " Pongo comilla simple ' dentro de doble ";
?>
```

```
Pongo comilla doble " dentro de simple. Pongo comilla simple ' dentro de doble
```

- ❖ Con ambos tipos de comillas podemos escribir en varias líneas el texto.

```
<?php
echo ' Escribo en
varias líneas con
comillas simples. ';
echo " Escribo en
varias líneas con
comillas dobles. ";
?>
```

```
Escribo en varias líneas con comillas simples. Escribo en varias líneas con comillas dobles.
```

Las cadenas son una sucesión de caracteres en un orden determinado, por lo que podemos acceder a parte de los caracteres, y lo hacemos usando corchetes [] y un índice numérico correspondiente a la posición del carácter buscado dentro del string.

```
$cadena= "hola mundo";
echo "la primera letra de mi variable es $cadena[0]";
```

```
la primera letra de mi variable es h
```

Ejemplo string

4.6.5 Array

Los arrays o matrices son estructuras que permiten el almacenamiento de un conjunto de datos bajo un mismo nombre.

Es un conjunto ordenado de elementos identificados por un índice, de modo que cada posición marcada por el índice contiene un valor. Lo que hace es asociar valores con claves.

La longitud del array se modifica dinámicamente cuando se añade o elimina un elemento.

Pueden estar compuestos por elementos de diferente tipo y la posición del primer elemento es la **0**.

Tipos:

- Arrays indexados o escalares: Arrays con índice numérico.
- Arrays asociativos: Arrays que utilizan como índice una cadena de caracteres. Se accede al contenido (valor) a través de la clave (índice).
- Arrays multidimensionales: Arrays cuyo contenido es uno o más arrays y necesitan varios índices para el acceso a los valores.

Array escalar			
Índice	0	1	2
Valor	Programación	Bases de Datos	FOL

Array asociativo			
Índice	PR	BD	FL
Valor	Programación	Bases de Datos	FOL

Array multidimensional						
Índice	0		1		2	
Índice	0	1	0	1	0	1
Valor	Programación	256	Bases de datos	160	FOL	96

4.6.5.1 Array escalar

Un array escalar, indexado, o simple, está formado por un conjunto de valores ordenados respecto a un índice de tipo entero. El índice indica la posición del elemento. El primer índice puede ser cualquier número, pero se recomienda empezar en cero. Es lo que conocemos como vector.

Hay dos formas de crear un array indexado:

- Utilizando la función "array()".

```
$signaturas = array("Programación", "Bases de datos", "FOL");
```

```
$signaturas = array(
    0=>"Programación",
    1=>"Bases de datos",
    2=>"FOL");
```

- Asignando directamente el contenido. Se puede indicar o no el índice.

```
$signaturas = ["Programación", "Bases de datos", "FOL"];
```

```
$signaturas[0] = "Programación";
$signaturas[1] = "Bases de datos";
$signaturas[2] = "FOL";
```

```
$signaturas[] = "Programación";
$signaturas[] = "Bases de datos";
$signaturas[] = "FOL";
```

Independientemente de cómo se haya creado, cuando se asigna un valor a una variable array usando corchetes vacíos, el valor se añadirá al final del array.

4.6.5.2 Array asociativo

Un array asociativo es aquel que utiliza como índices cadenas de caracteres.

Hay dos formas de crear un array asociativo:

- Utilizando la función “array()”.

```
$asignaturas = array(
    "PR"=>"Programación",
    "BD"=>"Bases de datos",
    "FL"=>"FOL");
```

- Asignando directamente el contenido.

```
$asignaturas["PR"] = "Programación";
$asignaturas["BD"] = "Bases de datos";
$asignaturas["FL"] = "FOL";
```

4.6.5.3 Array multidimensional

Un array multidimensional es la combinación de arrays unidimensionales (que pueden ser tanto de tipo escalar como asociativo).

Cada variable del array puede ser un array, y así creamos arrays de tres o más dimensiones.

Para manejar el array necesitamos un índice por cada una de las dimensiones que tiene.

Podemos usar la función array() para crearlos.

```
$asignaturas = array(
    array("Programación", 256),
    array("Bases de datos", 160),
    array("FOL", 96)
);
```

Ejemplo Array multidimensional

Estandar de codificación:

- No se permiten los números negativos como índices.
- Los índices en array indexados pueden iniciarse con cualquier valor no negativo, pero se recomienda iniciarlos a cero.
- Para mejorar la legibilidad, se debe agregar un espacio después de cada coma de separación de los índices en la declaración del array.

- Se recomienda hacer la declaración de los arrays asociativos divididos en diferentes líneas.
- Se recomienda dejar espacio antes y después de los operadores de asignación “=>” y de forma que queden alineados.

Ejercicio 2

4.6.5.4 list()

Es una instrucción asigna valores a una lista de variables en una sola sentencia.

```
$info = array('café', 'marrón', 'cafeína');
list($drink, $color, $power) = $info; // Listando todas las variables
echo "El $drink es $color y la $power lo hace especial.\n";
list($drink, , $power) = $info; // Listando algunas variables
echo "El $drink tiene $power.\n";
list( , , $power) = $info; // Mostramos sólo la tercera
echo "Necesito $power!\n";
```

Más sobre list()

4.6.6 Object

Un objeto es un tipo de dato que almacena tanto datos como los procedimientos necesarios para trabajar con ellos. Los objetos deben ser declarados de forma explícita, con una clase del objeto. Una clase es una estructura que contiene propiedades y métodos para el adecuado uso de los objetos. A partir de ella instanciamos objetos con esas mismas propiedades y métodos. Lo vemos más adelante.

```
class Coche {
    var $marca;
    function Coche() {
        $this->marca = "SEAT"; //define la marca del coche
    }
}

$panda = new Coche();// crear un objeto
echo $panda->marca;// imprime el valor de la propiedad marca: SEAT
```

4.6.7 Resource

Las variables de tipo resource no son en realidad un tipo de dato si no que son variables especiales que contiene una referencia a un recurso externo. Los recursos son creados y usados por funciones especiales. Lo veremos más adelante con bases de datos o acceso a ficheros.

4.7 Funciones para variables

Funciones para trabajar con variables:

- `gettype (variable)` : Devuelve el tipo de dato de una variable.
- `settype (variable, tipo)`: Estable el tipo de dato a una variable. Devuelve true o false según se haya podido hacer o no.
- `isset (variable)`: Devuelve true si la variable está definida y no es NULL. Si son pasados varios parámetros, devolverá TRUE únicamente si todos los parámetros están definidos.
- `unset (variable)`: Destruye las variables especificadas liberando así la memoria que ocupaban.

Ejemplo isset y unset

- `empty (variable)`: Devuelve true si la variable no se ha inicializado o es cero, false o cadena vacía.
- `is_int (variable)`, `is_integer (variable)`, `is_long (variable)`: Devuelve true si la variable es entero
- `is_float (variable)`, `is_real (variable)`, `is_double (variable)`: Devuelve true si la variable es float
- `is_numeric (variable)`: Devuelve true si la variable es un número
- `is_bool (variable)`: Devuelve true si la variable es de tipo lógico
- `is_array (variable)`: Devuelve true si la variable es array
- `is_string (variable)`: Devuelve true si la variable es string
- `is_object (variable)`: Devuelve true si la variable es objeto
- `var_dump (variable)`: Muestra información estructurada sobre una o más expresiones, incluyendo su tipo y valor. Las matrices y los objetos se muestran recursivamente para mostrar toda su estructura

```

1 <?php
2 $b = 3.1;
3 $c = true;
4 var_dump($b, $c);
5 ?>

```

localhost/pruebas/dos.php

float(3.1) bool(true)

- `print_r (variable)`: Muestra información legible sobre una variable. Es similar a `var_dump`, aunque únicamente se puede poner una variable.

```

1 <?php
2 $a = array ('a' => 'manzana', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
3 var_dump($a);
4 echo " --- -- ";
5 print_r ($a);
6 ?>

```

localhost/pruebas/dos.php

array(3) { ["a"]=> string(7) "manzana" ["b"]=> string(6) "banana" ["c"]=> array(3) { [0]=> string(1) "x" [1]=> string(1) "y" [2]=> string(1) "z" } } --- -- Array ([a] => manzana [b] => banana [c] => Array ([0] => x [1] => y [2] => z))

Ejemplo var_dump y print_r

5. Estructuras de Control

Son instrucciones orientadas a controlar el flujo de ejecución de los programas y que se basan en conceptos algorítmicos de gran funcionalidad.

Permiten bifurcar el flujo del programa y así ejecutar unas partes u otras de código (bloques de código) según se cumplan una serie de condiciones, hacer que un determinado código no llegue a ejecutarse nunca o que lo haga las veces que queramos.

Bloque de código: Conjunto de instrucciones agrupado entre llaves: { }. Un bloque de una única instrucción no necesita ir entre llaves.

PHP ofrece una sintaxis alternativa para alguna de sus estructuras de control: if, while, for, switch. En la sintaxis alternativa se cambia la llave de apertura { por dos puntos : y la de cierre } por endif;, endwhile;, endfor;, o endswitch;.

5.1 Estructuras de selección (Sentencias condicionales)

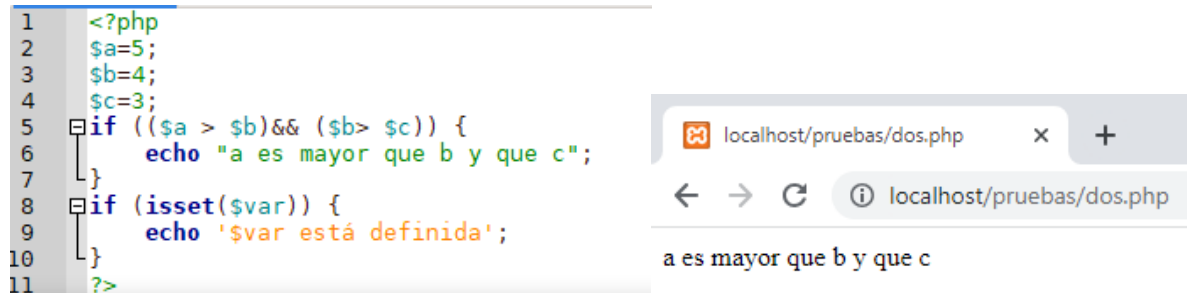
Son un conjunto de instrucciones que nos van a permitir ejecutar un grupo u otro de sentencias dependiendo de una o más condiciones.

5.1.1 If

if condicion { sentencias; }	if condicion: sentencias; endif ;	<u>Operador ternario o If compacto:</u> condicion ? sentencia1 : sentencia2
if condicion { sentencias1; } else { sentencias2; }	if condicion { sentencias1; } else { if condicion2 { sentencias2; } }	if condicion { sentencias1; } elseif condicion2 { sentencias2; } else { sentencias3; }

Las condiciones que es capaz de evaluar la instrucción if puede ser cualquier tipo de expresión, lo que significa que podría contener varias igualdades o desigualdades e incluso el resultado devuelto por alguna función.

Las sentencias if pueden anidarse dentro de otras sentencias if infinitamente.



```

1  <?php
2  $a=5;
3  $b=4;
4  $c=3;
5  if (($a > $b)&& ($b > $c)) {
6      echo "a es mayor que b y que c";
7  }
8  if (isset($var)) {
9      echo '$var está definida';
10 }
11 ?>

```

localhost/pruebas/dos.php x +

localhost/pruebas/dos.php

a es mayor que b y que c

Estandar de codificación:

- Las sentencias de control basadas en if y elseif deben tener un único espacio en blanco antes del paréntesis inicial y un único espacio en blanco después del paréntesis final.
- Los operadores dentro de la condición deben separarse con espacios. Es recomendable usar paréntesis internos para mejorar la agrupación de las expresiones condicionales largas.
- La llave de inicio { debe estar en la misma línea que la condición, mientras que la de cierre } debe estar sola. El código dentro de las llaves debe estar indentado:

```

...
if ($a != 2) {
    $a = 2;
}
...

```

- Si el largo de la condición es mayor al largo máximo de la línea, y la condición tiene varias cláusulas, se puede partir en varias líneas:

```

...
if (($a == $b)
    && ($b == $c)
    || (Foo::CONST == $d)
){
    $a = $d;
}
...

```

- Toda sentencia if, elseif o else debe usar llaves de apertura { y cierre } sin excepción.

Ejemplo If

Ejercicio 3

5.1.2 Switch

Se usa para comparar la misma variable (o expresión) con varios valores diferentes, y ejecutar una parte de código distinta dependiendo de a qué valor sea igual.

```
switch (expresión){
    case valor1: instrucciones1; [break;]
    case valor2: instrucciones2; [break;]
    ...
    case valorN: instruccionesN; [break;];
    [default: instruccionesN+1;]
}
```

El siguiente código:

```
<?php
$i=0;
switch ($i) {
    case 0: echo "i es igual a 0 ";
    case 1: echo "i es igual a 1 ";
    case 2: echo "i es igual a 2 ";
        break;
    default: echo "i no es igual a 0, 1 ni 2 ";
}
?>
```

Mostrará en el navegador:

```
i es igual a 0 i es igual a 1 i es igual a 2
```

La sentencia switch ejecuta línea por línea (sentencia por sentencia). Cuando una sentencia case es encontrada con un valor que coincide con el valor de la sentencia switch, PHP comienza a ejecutar las sentencias. PHP continúa ejecutando las sentencias hasta el final del bloque switch, o hasta la primera vez que vea una sentencia break. Si no se escribe una sentencia break al final de la lista de sentencias de un caso, PHP seguirá ejecutando las sentencias del caso siguiente.

La instrucción break tiene por misión finalizar el flujo de la secuencia, es decir, dar por terminada la ejecución de la instrucción switch.

La sentencia "case" no lleva { } porque cada "case" se finaliza con el "break".

La sentencia default se utiliza para el caso en el que la expresión evaluada no coincida con ninguna de las sentencias case.

Estandar de codificación:

- El código dentro de un bloque switch y dentro de cada case debe estar indentado.

- Si se omite intencionalmente un break dentro de un case, debe escribirse un comentario justificando el motivo.

5.2 Estructuras repetitivas o de recorrido (Sentencias de bucle)

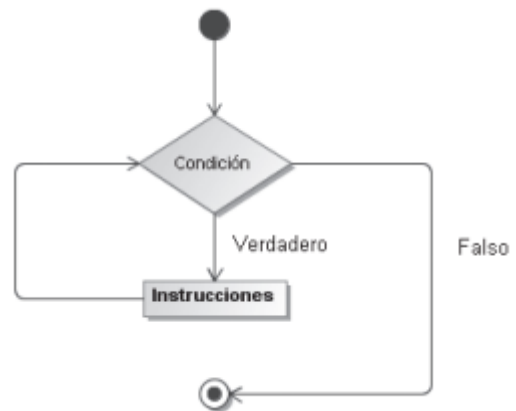
Los bucles permiten repetir la ejecución de un grupo de sentencias dependiendo de una o más condiciones.

Es importante tener siempre bajo control la condición a evaluar para evitar caer en un bucle infinito, es decir, una ejecución de sentencias repetida sin condición de parada. En PHP hay tres ciclos, que en realidad pueden hacer las mismas cosas, pero podemos optar por uno u otro según las circunstancias o el problema a solucionar.

4.2.1 While

Se usa para ejecutar un grupo de sentencias mientras se cumpla una condición.

En cada iteración del bucle se evalúa la condición y si esta es verdadera pasan a ejecutarse las instrucciones contenidas en el cuerpo del bucle. El bucle termina cuando el resultado de evaluar la condición es falso, es decir, cuando la condición ha dejado de cumplirse.



```
while (condición){
    instrucciones;
}
```

```
while (condición):
    instrucciones;
endwhile;
```

5.2.2 Do-While

Se usa para ejecutar un grupo de sentencias hasta que se deje de cumplir una condición.

Funciona igual que el bucle while, excepto que la condición se comprueba cuando se ha realizado una iteración. Esto hace que al menos el cuerpo del bucle se realiza una vez, aunque la expresión de la condición se evalúe a false.



```
do{
    instrucciones;
} while (condición);
```

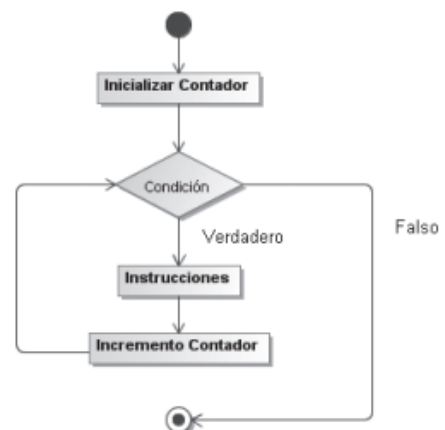
[Ejercicio 4](#)

5.2.3 For

Su usa para ejecutar un grupo de sentencias un número determinado de veces.

Es capaz de recorrer estructuras de forma repetitiva desde un valor inicial hasta otro final según un incremento.

Se pueden usar expresiones vacías en los bucles for y el incremento también puede ser no unitario.



```
for ([inicializacion] ; [condición] ; [incremento])
    instrucciones;
};
```

```
for ([inicializacion] ; [condición] ; [incremento]):
    instrucciones;
endfor;
```

- Inicialización: se suele usar para inicializar y declarar la variable o variables que se van a utilizar como contador del bucle. Sólo se ejecuta una vez al principio del bucle. Si hay más de una variable se separan por comas.
- Condición: define las condiciones que deben cumplirse para poder ejecutar las sentencias del bucle. La expresión se evalúa antes de cada iteración.
- Incremento: modifica el valor de la variable o variables utilizadas como contadores del bucle.

/* ejemplo 1 */


```

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}
/* ejemplo 2 */
for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}
/* ejemplo 3 */
$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}
/* ejemplo 4 */
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);

```

Ejemplo For con Array

Ejemplo For con Matriz (array con array)

[Ejercicio 5](#)

[Ejercicio 6](#)

5.2.4 Foreach

Permite recorrer las estructuras de tipo array de una forma sencilla. Funciona sólo sobre arrays, y emitirá un error al intentar usarlo con una variable de otro tipo o una variable no inicializada. No requiere ni expresión de finalización ni de incremento ya que foreach se limita a recorrer los elementos del array de uno en uno hasta el último

(a menos que indiquemos otro fin dentro del bucle). Cuando foreach inicia su ejecución, el puntero interno del array se pone automáticamente en el primer elemento del array.

foreach (expr_array as \$valor) { sentencias;}	foreach (expr_array as \$clave => \$valor) { sentencias;}
--	---

La primera forma recorre el array dado por expr_array. En cada iteración, el valor del elemento actual se asigna a \$valor y el puntero interno del array avanza una posición (así en la próxima iteración se estará observando el siguiente elemento).

La segunda forma además asigna la clave del elemento actual a la variable \$clave en cada iteración. Así hacemos lo mismo que con la anterior pero nos permite conocer la posición exacta (el índice) del componente actual dentro del array.

/ ejemplo 1: sólo valor*/*

```
$vector = array(1, 2, 3, 17);
```

```
foreach($vector as $valor) {
```

```
    echo "Valor actual de \$vector: $valor.<br>";
```

```
}
```

*/*Hacer esto con un bucle for */*

```
for($i=0;$i<=count($vector);$i++) {
```

```
    echo "Valor actual de \$vector: $vector[$i].<br>";
```

```
}
```

/ ejemplo 2: clave y valor */*

```
$vector = array(
```

```
    "uno" =>1,
```

```
    "dos" =>2,
```

```
    "tres" =>3,
```

```
    "diecisiete" =>17
```

```
);
```

```
foreach($vector as $k => $v) {
```

```
    echo "\$vector[$k] => $v.<br>";
```

```
}
```

/ ejemplo 3: matriz multi-dimensional */*

```
$a[0][0] = "a";
```

```
$a[0][1] = "b";
```

```

$a[1][0] = "y";
$a[1][1] = "z";
foreach($a as $v1) {
    foreach ($v1 as $v2) {
        echo "$v2<br>";
    }
}

```

Ejemplo Foreach Array escalar

Ejemplo Foreach Array asociativo

Ejemplo Foreach Array asociativo multidimensional

5.3 Instrucciones para ruptura, finalización, salto y retorno

PHP tiene instrucciones pertenecientes a la categoría de estructuras de control pero que no se usan para repetir o recorrer variables, sino que se usan para terminar o escapar de una iteración y devolver valores o el control de flujo a otra parte del código.

5.3.1 Break

Además de para la sentencia switch, se utiliza para finalizar un bucle antes de que sus condiciones no se cumplan.

Break puede llevar un argumento numérico que indica de cuantas estructuras anidadas debe salir.

```

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

```

```

        break;
    }
}

```

5.3.2 Continue

Para saltar a la iteración siguiente. Al alcanzar esa sentencia, dentro de un bucle, las sentencias restantes no se ejecutan y se vuelve a evaluar de nuevo la expresión del bucle continuando con la siguiente iteración.

```

$vector = array (1, 2, 3, NULL, 4, 5);
foreach($vector as $valor) {
    if ($valor==NULL){
        continue;
    }
    print "Valor actual: $valor.\n";
}

```

*/*Recorre \$vector hasta que encuentra el valor nulo, cuya impresión se salta o sea que imprime: 1, 2, 3, 4, 5*/*

Igual que Break, acepta un argumento numérico opcional, que indica a cuántos niveles de bucles encerrados se ha de saltar al final. El valor por omisión es 1, por lo que salta al final del bucle actual.

6. Funciones

Son agrupaciones de instrucciones con una funcionalidad concreta a las que podemos invocar desde nuestro código.

Las funciones dividen las tareas que debe hacer un script, agrupando instrucciones relacionadas para la ejecución de una tarea.

Sintaxis:

```
Function nombre (parámetro1, parámetro2, ..., parámetroN) {  
    Instrucciones;  
}
```

Las funciones pueden recibir valores desde las sentencias que las llaman (parámetros o argumentos) y pueden devolver valores. Los parámetros se usan como variables locales dentro de la función.

Se pueden invocar a las funciones de varias formas, como por ejemplo pasando su resultado a una variable (u otra función), en una condición de un bucle, en una sentencia "echo" o incluso escribiendo su nombre directamente.

PHP cuenta con dos tipos de funciones: las creadas por los usuarios y las incluidas en el propio sistema.

Los nombres de funciones se pueden llamar con mayúsculas o minúsculas, aunque es una buena costumbre llamar a las funciones tal y como aparecen en su definición. Una práctica bastante común a la hora de bautizar a una función (y también más adelante a los métodos) será mediante el estilo camelCase, es decir, juntar varias palabras que definen su funcionalidad con la primera letra de cada una en mayúsculas salvo la primera (crearCabecera, calcularFactura, etc).

Se pueden llamar desde cualquier lugar del fichero de php en el que se han creado (pueden definirse antes o después de usarse), y también se puede importar el fichero a otro para poder llamarla desde el otro.

Si hay una función dentro de otra función se podrá llamar únicamente desde la que la engloba, no desde fuera.

6.1 Devolver valores

Una función puede devolver un resultado mediante la instrucción `return ()` o no hacerlo porque su cometido se lleva a cabo dentro de la propia función.

Función definida por el usuario que devuelve valor:

```
Function creaCabecera($titulo) {
    return "<html><head><title>$titulo</title></head>";
}
```

Llamada a la función:

```
<?php
    echo creaCabecera("mi web de ejemplo"); //invocamos a la función
¿>
```

Función definida por el usuario que NO devuelve valor:

```
Function creaCabecera($titulo) {
    echo "<html><head><title>$titulo</title></head>";
}
```

Llamada a la función:

```
<?php
    creaCabecera("mi web de ejemplo"); //invocamos a la función
¿>
```

La instrucción return devuelve sólo un valor, para devolver más de uno los podemos encapsular dentro de un dato estructurado como un array o un objeto.

6.2 Argumentos de funciones

En PHP los argumentos o parámetros de las funciones se pasan por **valor**, por lo que si el valor del argumento dentro de la función se cambia, no se cambia fuera de la función. Para permitir a una función modificar sus argumentos, éstos deben pasarse por referencia.

Para hacer que un argumento a una función sea siempre pasado por **referencia** hay que poner delante del nombre del argumento el signo 'ampersand' (&) en la definición de la función.

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

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

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

Ejemplo paso parámetros

6.2.1 Argumentos predeterminados

Los argumentos predeterminados son argumentos que toman un valor por defecto. Se pueden definir para argumentos escalares y debe ser una expresión constante, no una variable, un miembro de una clase o una llamada a una función.

Al usarse argumentos predeterminados, éstos se ponen a la derecha de los argumentos no predeterminados.

```
function hacercafé($tipo = "capuchino")
```

```
{
```

```
    return "Hacer una taza de $tipo.\n";
```

```
}
```

```
echo hacercafé();
```

```
echo hacercafé(null);
```

```
echo hacercafé("espresso");
```

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

Ejemplo argumento predeterminado

6.3 Ámbito de las variables

El ámbito de una variable es el contexto dentro del que la variable está definida.

6.3.1 Variables locales

Se encuentran definidas en un entorno cerrado, como por ejemplo, en una función. Fuera de ese entorno no existen.

Si queremos obtener ese valor para usarlo fuera, lo podemos devolver en la instrucción `return()` y asignarlo así a otra variable o imprimirlo directamente con `echo`.

```
<?php
```

```
Function creaNumero() {
```

```
    $num=rand ();
```

```
    Return ($num);
```

```
}
```

```
Echo creaNumero();
```

```
¿>
```

Ejemplo devolución variables locales

6.3.2 Variables globales

Están definidas fuera de las funciones y las funciones no tienen acceso a ellas, a no ser que se especifique usando la palabra reservada **global** o usando el array **\$GLOBALS**.

```
<?php
$num=8;
Function creaNumero() {
    $num=24;
    echo $num; //imprime 24, no se puede usar la definida fuera
}
creaNumero();
Echo $num; //imprime 8, ya que la de la función no modifica a la global
¿>
```

```
<?php
$a = 1;
$b = 2;
function Suma()
{
    global $a, $b; //tiene acceso a las variables definidas globalmente
    $b = $a + $b;
}
Suma();
echo $b; //imprime 3 porque la variable se ha modificado en la función
?>
```

```
<?php
$a = 1;
$b = 2;
function Suma()
```



```

{
    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}
Suma();
echo $b; //imprime 3 porque la variable se ha modificado en la función
?>

```

Ejemplo variables globales

Ejemplo variables globales con array

6.3.3 Variables estáticas

Existen sólo en el ámbito local de la función, pero no pierde su valor cuando la ejecución del programa abandona este ámbito.

```

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

```

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

6.4 Recursividad

Una función recursiva es aquella que se llama a sí misma.

6.5 Uso de librerías de funciones

Una librería, o biblioteca, es un conjunto de funciones u objetos que poseen algo en común y una interface bien definida que se localizan en un archivo externo a nuestro proyecto. Mejora la estructuración del código y permite que ciertos proyectos se ayuden del trabajo de otros programadores.

Para crear una librería, programar el código de forma normal con sus correspondientes etiquetas de apertura y cierre y guardamos el archivo con extensión **php** o **inc**. Después debemos usar desde la página principal las instrucciones **include** o **require**.

```
<?php
include "lib/MisFunciones.php";
include "bbdd.inc";
require "seguridad.php";
¿>
```

La instrucción **include** es más tolerante a fallos que **require** y permitirá continuar con la ejecución del programa aunque se encuentre un fallo. Típicamente se hace uso de **require** para aplicaciones que necesiten obligatoriamente algún archivo crítico.

Cuando necesitamos incluir muchas librerías, puede ser que, por error, incluyamos el mismo archivo en más de una ocasión, generándose a menudo errores de difícil detección. Para evitarlo podemos usar las variantes **include_once** y **require_once**.

Cuando se incluye un archivo, el código que contiene hereda el ámbito de las variables de la línea en la cual ocurre la inclusión. Cualquier variable disponible en esa línea del archivo que hace el llamado, estará disponible en el archivo llamado, desde ese punto en adelante. Sin embargo, todas las funciones y clases definidas en el archivo incluido tienen el ámbito global.

vars.php

```
<?php
$color = 'verde';
$fruta = 'manzana';
?>
```

test.php

```
<?php
echo "Una $fruta $color"; // Una
include 'vars.php';
echo "Una $fruta $color"; // Una manzana verde
?>
```

El desarrollo de páginas web a menudo también se vale de las instrucciones **include** y su funcionalidad para construir páginas de forma modular, algo especialmente útil en caso de que existan secciones que se deben repetir. Es fácil encontrarse, por

ejemplo, con webs cuyas primeras líneas de código (encabezado y los primeros elementos como logotipo, menús, etc) residen en un archivo y el cuerpo con los contenidos principales en otro archivo distinto que será el que incluya los demás módulos:

```
<html>
<head>
    <?php include ("header.php");?>
</head>
<body>
    <?php include ("menus.php"); ?>
    <div id="cuerpo">
        ...
    </div>
    <?php
        include ("sideNav.php");
        include ("bottom.php");
    ¿>
</body>
</html>
```

6.6 Funciones para tratamiento de texto y arrays

6.6.1 Funciones para el tratamiento de arrays.

- **In_array**: comprueba si un valor existe en un array. Devuelve True o False. Es case sensitive.
- **Count**: cuenta los elementos de una matriz.
- **Unset**: al igual que con las variables, podemos usar esta función para eliminar elementos de un array.

Para poder eliminar un elemento en concreto deberemos conocer su índice. De lo contrario, si llamamos a la función sin índices hacemos vaciaremos todo el array

```
$vector = array(0, 10, 20, 30);
```

```

var_dump($vector);
unset($vector [2]);
echo "<br>";
var_dump($vector);
array(4) { [0]=> int(0) [1]=> int(10) [2]=> int(20) [3]=> int(30) }
array(3) { [0]=> int(0) [1]=> int(10) [3]=> int(30) }

```

- **Sort:** Ordena un array.
- **Rsort:** Ordena un array en orden inverso.
- **current():** Devuelve el elemento actual en un array.
- **next():** Devuelve el siguiente valor del array y avanza el puntero interno del array un lugar.
- **prev():** Se comporta como next(), a excepción de que rebobina el puntero interno del array una posición en lugar de avanzar.
- **reset():** Rebobina el puntero interno de un array al primer elemento y devuelve el valor del primer elemento del array.

```

$transporte = array('pie', 'bici', 'coche', 'avión');
$modo = current($transporte); // $modo = 'pie';
$modo = next($transporte); // $modo = 'bici';
$modo = current($transporte); // $modo = 'bici';
$modo = prev($transporte); // $modo = 'pie';
$modo = reset($transporte); // $modo = 'pie';

```

- **Key:** Devuelve la clave del elemento del array que está apuntando actualmente el puntero interno. No desplaza el puntero. Si el puntero interno señala más allá del final de la lista de elementos o el array está vacío, key() devuelve NULL.

```

$array = array(
    'fruta1' => 'manzana',
    'fruta2' => 'naranja',
    'fruta3' => 'uva',
    'fruta4' => 'manzana',
    'fruta5' => 'manzana');

// Muestra las claves del array donde el valor equivale a "manzana"
while ($nombre_fruta = current($array)) {
    if ($nombre_fruta == 'manzana') {
        echo key($array). "<br />";
    }
}

```

```

    next($array);
}

```

- **array_merge**: combina los elementos de uno o más arrays juntándolos de modo que los valores de uno se anexan al final del anterior. Retorna el array resultante. Si los arrays de entrada tienen las mismas claves de tipo string, el último valor para esa clave sobrescribirá al anterior. Sin embargo, los arrays que contengan claves numéricas, el último valor no sobrescribirá el valor original, sino que será añadido al final. Los valores del array de entrada con claves numéricas serán renumeradas con claves incrementales en el array resultante, comenzando desde cero.

```

$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green",
"shape" => "trapezoid",4);
$resultado = array_merge($array1, $array2);
print_r($resultado);

```

```

Array
(
    [color] => green
    [0] => 2
    [1] => 4
    [2] => a
    [3] => b
    [shape] => trapezoid
    [4] => 4
)

```

[Ejercicio 7a D.N.I.](#)

6.6.2 Funciones para el tratamiento de cadenas de texto

La mayoría de las funciones orientadas al tratamiento de cadenas de texto están pensadas para tomar una cadena como argumento y devolver algún dato (longitud, posición de un carácter, etc), otra cadena procesada o incluso un array.

- **strlen (\$cadena)**: longitud de una cadena
- **strpos (\$cadena, \$letra)**: encuentra la posición de la primera ocurrencia de un carácter dentro de una cadena; Es sensible a mayúsculas y minúsculas

```

$mystring = 'abc';
$findme = 'b';
$pos = strpos($mystring, $findme); //devolverá la posición 1 pero si la ponemos
en mayúsculas no la encuentra y no devuelve nada.

```

- **strcmp (\$cadena1, \$cadena2)**: Devuelve 0 si ambas cadenas son iguales.

```

if (strcmp ($cadena1, $cadena2)==0) {
    echo "ambas cadenas son iguales";
}

```

- **strstr(\$cadena, "texto")**: encuentra la primera aparición de una cadena dentro de otra cadena; no devuelve una posición sino el string a partir del punto en el que aparece el texto buscado.

```
$email = 'name@example.com';
$domain = strstr($email, '@');
echo $domain; // mostrará @example.com
```

- **substr (\$cadena, \$inicio, \$longitud)**: devuelve parte de una cadena desde \$inicio y con longitud \$longitud. Si \$inicio es un entero negativo empezará por el final de la cadena. Si \$longitud no se indica la función devolverá todo el texto desde \$inicio

```
$resto = substr("abcdef", 1, 1); // devuelve "b"
$resto = substr("abcdef", 2); // devuelve "cdef"
$resto = substr("abcdef", -1); // devuelve "f"
$resto = substr("abcdef", -2); // devuelve "ef"
$resto = substr("abcdef", -3, 1); // devuelve "d"
```

- **str_replace (\$texto1, \$texto2, \$cadena)**: busca todas las apariciones del texto \$texto1 en \$cadena y lo sustituye por \$texto2.
- **str_shuffle (\$cadena)**: reordena aleatoriamente el contenido de la cadena de entrada y devuelve una cadena de la misma longitud pero con sus caracteres reordenados.

```
$cadena = 'abcdef';
$desordenada = str_shuffle($cadena);
echo $desordenada; // Esto imprime algo como: bfdaec
```

- **htmlspecialchars (\$cadena)**: esta función se suele usar para recoger datos desde un campo de formulario para garantizar que cualquier carácter que sea especial en html se codifique adecuadamente, de manera que nadie pueda inyectar etiquetas HTML o Javascript en nuestras páginas y provocar errores.
- **rtrim (\$cadena)**: retira los espacios en blanco (u otros caracteres) del final de un string. También están **ltrim** y **trim** (elimina los de izquierda y derecha, no los interiores).

```
echo rtrim("Texto con espacios "); //imprime "Texto con espacios"
echo rtrim("Texto con punto.", "."); //imprime "Texto con punto"
```

- **strtolower (\$cadena)** y **strtoupper (\$cadena)**: convierten el contenido de un texto todo a minúsculas o toda a mayúsculas respectivamente

Ejercicio 8 Contraseña automática aleatoria

6.7 Otras funciones

- **header (“Location: http://.....” [,replace] [,http_response_code]):** redirige a un usuario de una página a otra página.

Hay que usar la función `exit()` o la función `die()` inmediatamente después de la redirección con `header` para detener la ejecución del script y evitar resultados no deseados.

```
header("Location: https://www.agilcentros.es/audio/index.php");  
exit;
```

- **exit ([“Mensaje”]):** Imprime un mensaje y termina el script actual. Los paréntesis se pueden omitir si no se muestra ningún mensaje (que sería mostrado antes de terminar).

7. Paso de información entre documentos

Una de las funcionalidades de PHP es la de pasar información de una página a otra a medida que un usuario las visita. PHP proporciona varias herramientas como son el pase de información a través de formularios y la cabecera http, el uso de cookies y el tratamiento de sesiones.

7.1 Paso de información con formularios: GET y POST

Una de las fórmulas más empleadas para transmitir información de una página a otra es mediante el uso de formularios y el protocolo HTTP. Por ejemplo, al pulsar sobre un enlace o botón, se está enviando información al servidor, la cual será procesada y provocará una respuesta de algún tipo. También al rellenar un formulario, la información incluida se envía a algún lugar para ser tratada.

Para este tipo de operaciones PHP usa dos métodos: GET y POST.

Ambos métodos funcionan de forma similar, haciendo uso de unas variables superglobales de tipo array asociativo (\$_GET y \$_POST) a las que podremos acceder en todo momento.

El atributo **method** del form especifica el método usado para enviar los datos del formulario:

- **get**: los datos del formulario se agregan (con el carácter “?” como separador) al URL especificado en el atributo action (el programa que tratará los datos).
`http://localhost/php/accion.php?nombre=Jose&edad=22`
- **post**: los datos del formulario se agregan al cuerpo del mensaje, y por lo tanto viajan ocultos.

Ejemplo GET:

```
<form action="accion.php" method="get">
<p>Su nombre: <input type="text" name="nombre" /></p>
<p>Su edad: <input type="text" name="edad" /></p>
<p><input type="submit" /></p>
</form>
```

Al pulsar el botón de submit, el formulario HTML enviará los valores introducidos en “nombre” y “edad” al archivo accion.php mediante un el método GET.

En el archivo accion.php asociado:

Hola <?php echo htmlspecialchars(\$_GET['nombre']); ?>.

Usted tiene <?php echo (int)\$_GET['edad']; ?> años.

Al hacer submit se crea un array asociativo con los pares índice-valor con los campos del formulario correspondientes al nombre y al valor y lo ha almacenado en un array llamado **\$_GET**. Después, el documento accion.php ha accedido a los valores de la variable **\$_GET** a través de sus índices “nombre” y “edad” y los ha impreso.

En el caso del nombre, se hace uso de la función htmlspecialchars() para garantizar que cualquier carácter que sea especial en html se codifique adecuadamente y que nadie pueda inyectar etiquetas HTML o Javascript en la página. El campo edad, ya que es un número, podemos convertirlo a un valor de tipo integer que automáticamente se deshará de cualquier carácter no numérico.

Este mismo ejemplo podría ser usado con el método POST en el formulario y recogido igualmente desde accion.php mediante la variable superglobal **\$_POST**.

Diferencias entre los métodos GET y POST:

Con GET, todos los pares campo-valor son enviados a través de la URL en su llamada al archivo destino. Estas llamadas son visibles por cualquiera en tiempo de navegación y también pueden ser cacheadas (historial del navegador), indexadas por buscadores e incluso podemos agregar los enlaces a nuestros favoritos o hasta pasar una url completa a otra persona para que directamente ingrese a esa página. Con el método POST sin embargo no se puede hacer esto.

Es por ello que se tiende a pensar que el envío de información mediante el método GET es menos seguro que usando el método POST y que por lo tanto GET no debería ser usado. Sin embargo esto no es del todo cierto. Las solicitudes GET son sencillas, y no necesitan de un formulario, ya que este mecanismo nos permite crear vínculos que envíen directamente parámetros de una página a otra desde dentro del código.

Con el método POST, el navegador hace una conexión al servidor y se encarga de mandar los datos de manera separada, de esta forma los datos se mantienen hasta cierto punto privados, y tenemos la ventaja de que podemos enviar una mayor cantidad de datos en la solicitud.

El método POST debe ser utilizado en lugar de GET cuando deseamos transmitir mucha información, modificar datos en el servidor o en el caso de transmitir información sensible, como por ejemplo autenticar un usuario con contraseña, validar una compra, actualizar datos en una BBDD,...

El método GET debería usarse cuando el formulario no tiene efectos secundarios.

Ejemplo Formulario Get y Post

Ejercicio 7b D.N.I. con Formulario

7.2 Paso de información con COOKIES

PHP soporta el tratamiento de cookies HTTP para almacenar datos en el navegador del cliente y poder así monitorizar o identificar a usuarios que vuelven al sitio web.

Cookie: Fragmento de información que un navegador almacena en el disco duro del visitante de una página web. La información se almacena a petición del servidor y éste la puede recuperar en posteriores visitas del cliente a la misma página web.

Todo el tratamiento de las cookies se realiza en la cabecera del mensaje HTTP y , por lo tanto, deben ser manejadas antes de que se envíe cualquier otra información al navegador o nos dará un error (antes de la etiqueta <html>).

El valor de una cookie tiene preferencia sobre los valores pasados mediante un formulario, por lo que si un formulario y una cookie usan los mismos identificadores, los valores de la cookie sobrescribirán a los del formulario.

Algunas funciones son:

- Llevar el control de usuarios: cuando un usuario introduce su nombre de usuario y contraseña, se almacena una cookie para que no tenga que estar introduciéndolas para cada página del servidor.
- Conseguir información sobre los hábitos de navegación del usuario, e intentos de spyware (programas espía), por parte de agencias de publicidad y otros.

Estructura de las COOKIES

La información se almacena siguiendo una estructura de pares *identificador = valor*.

Además del nombre y el valor, pueden tener:

- caducidad (tiempo de validez)
- dominio (rango de dominios en los cuales es válida)
- ruta (directorio a partir del que la cookie es válida)
- seguro (la cookie será transmitida únicamente por un canal seguro).

Creación de cookies

bool **setcookie** (name, value[, expire][, path][, domain][, secure])

En la creación de cookies los únicos argumentos obligatorios son el nombre que se le asigna y su valor inicial.

Las cookies creadas serán incluidas automáticamente en el array asociativo **`$_COOKIE`**.

Hay que usar la función `setcookie()` antes de empezar a escribir el contenido de la página, porque si no PHP producirá un aviso y no se creará la cookie. El motivo es que las cookies se crean mediante cabeceras de respuesta HTTP y las cabeceras se envían antes del texto de la página. Es decir, cuando PHP encuentra una instrucción que escribe texto, cierra automáticamente la cabecera; si a continuación PHP encuentra en el programa la función `setcookie()`, da un aviso porque ya se han enviado las cabeceras y no se crea la cookie.

`Setcookie()` debe ser invocada antes de que cualquier otra salida sea enviada al navegador, antes de cualquier salida, incluyendo las etiquetas `<html>` y `<head>` así como cualquier espacio en blanco. Si existe salida antes de llamar esta función, `setcookie()` fallará y devolverá `FALSE`. Si `setcookie()` se ejecuta con éxito, devolverá `TRUE`.

Parámetros de `setcookie`:

Parámetro	Descripción	Ejemplos
name	Nombre de la cookie	'nombre_cookie' es llamada como <code>\$_COOKIE['nombre_cookie']</code>
value	Valor de la cookie. Almacenado en el cliente.	este valor es recuperado por medio de <code>\$_COOKIE['nombre_cookie']</code>
expire	Hora a la que xperia la cookie. Se puede definir con la función <code>time()</code> más los segundos	<code>time()+3600</code> expirará dentro de una hora a partir de su creación. <code>time()+60*60*24*30</code> expirará en 30 días. Si no se define, expirará al final de la sesión (cuando se cierre el navegador).
path	Ruta en el servidor en la que estará disponible la cookie.	Si se define como <code>'/'</code> , la cookie estará disponible en el dominio completo. Si se define como <code>'/foo/'</code> , la cookie será disponible únicamente al interior del directorio <code>/foo/</code> y todos sus subdirectorios. El valor predeterminado es el directorio actual en el que se define la cookie.

domain	Dominio en el que la cookie está disponible.	Para lograr que la cookie esté disponible en todos los subdominios de example.com será necesario definir este valor como '.example.com'.
secure	Indica que la cookie debería ser transmitida únicamente sobre una conexión HTTPS segura.	Cuando su valor es TRUE, la cookie será definida sólo si existe una conexión segura. El valor predeterminado es FALSE.

Si se desea asignar múltiples valores a una única cookie, solo se debe agregar [] al nombre de la cookie. Hay que crear cada elemento de la matriz en una cookie distinta.

//Definimos una única cookie de tipo array con dos valores

```
<?php
setcookie("datos[nombre]", "MiNombre");
setcookie("datos[apellidos]", "MiApellido");
?>
```

Setcookie () también nos permite actualizar el valor de una cookie si su nombre ya existía. Así podemos crear contadores de visitas.

Modificación de cookies

Para modificar una cookie ya existente, se debe volver a crear la cookie con el nuevo valor.

Eliminación de cookies

Para borrar una cookie:

-se puede volver a crear la cookie con un tiempo de expiración anterior al presente.

```
<?php
setcookie("nombre", "Valor", time() - 60); // Esta cookie se borrará inmediatamente.
?>
```

-se puede usar la misma función que para crearla, aunque únicamente con el nombre de la cookie a eliminar.

```
<?php
setcookie("nombre"); // Esta cookie no se borra, pero no guardará ningún valor.
?>
```

Consultar el contenido de cookies

Para acceder al contenido de las cookies tenemos la variable superglobal de tipo array asociativo **\$_COOKIE**, en la que el nombre de cada cookie es el índice.

\$_COOKIE ["nombreCookie"]

Mostrar el contenido de todas las cookies:

```
print_r ($_COOKIE);
```

Hay que tener en cuenta el orden en que se realiza el envío y la creación de cookies, así como su disponibilidad en **\$_COOKIES**:

- cuando una página pide al navegador que cree una cookie, el valor de la cookie no está disponible en **\$_COOKIE** en esa página.
- el valor de la cookie estará disponible en **\$_COOKIE** en páginas posteriores, cuando el navegador las pida y envíe el valor de la cookie en la petición.

Ejemplo:

```
<?php
setcookie("nombre", "Mi Nombre");

if (isset($_COOKIE["nombre"])) {
    print "<p>Su nombre es $_COOKIE[nombre]</p>\n";
} else {
    print "<p>No sé su nombre.</p>\n";
}
?>
```

La primera vez que se ejecuta este programa:

- 1.- el navegador pide la página, pero no envía con la petición el valor de ninguna cookie, porque la cookie todavía no existe
- 2.- el servidor envía la página: en la cabecera de respuesta:
 - 2a.- el servidor incluye la petición de creación de la cookie
 - 2b.- en la página escribe "No sé su nombre" porque no ha recibido ninguna cookie del navegador.

La segunda vez (y las siguientes) que se ejecuta este programa:

- 1.- el navegador pide la página y envía con la petición el valor de la cookie "nombre".
- 2.- el servidor envía la página:
 - 2a.- en la cabecera de respuesta, el servidor incluye la petición de creación de la cookie (como es el mismo valor, la cookie se queda igual).

2b.- en la página escribe "Su nombre es Mi Nombre" porque ha recibido la cookie del navegador.

Comprobar si las cookies están habilitadas en el navegador del cliente

Si un usuario desactiva el uso de cookies en su navegador y si nuestro sitio web requiere usar cookies para su correcto funcionamiento, deberíamos poder comprobarlo.

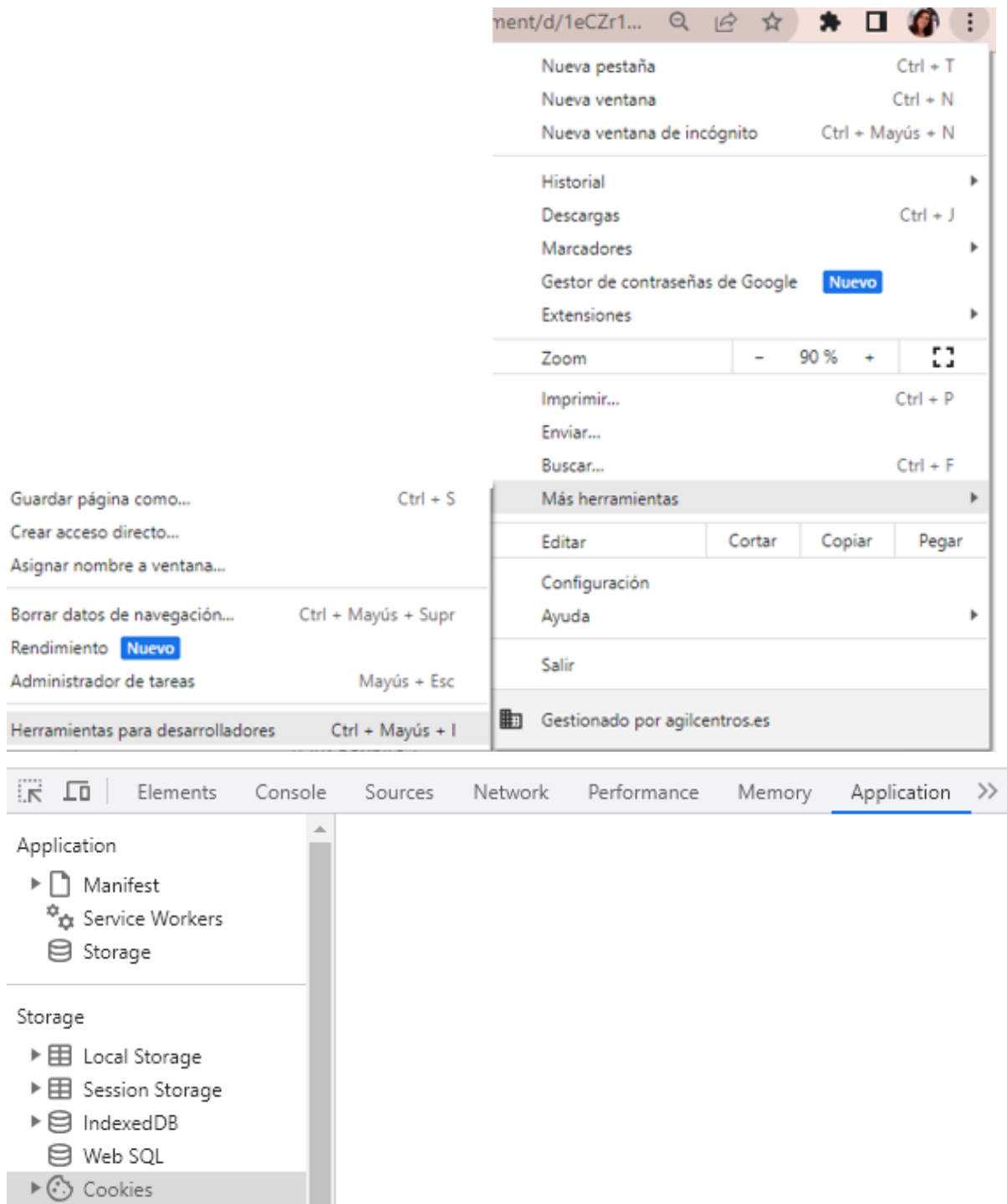
Una forma es probando a crear una cookie y acto seguido comprobar si podemos acceder a ella o comprobando si la variable \$_COOKIE contiene algún valor (mediante la función count()):

```
if(count($_COOKIE) > 0) {  
    echo "Cookies activadas.";  
} else {  
    echo "Cookies no activadas.";  
}
```

Ver las cookies en Chrome



-> Más herramientas -> Herramientas para desarrolladores -> Application -> Storage -> Cookies



Ejemplo Cookies

Ejercicio 9 Cookies

7.3 Paso de información con SESIONES

Una sesión es un mecanismo de programación que permite conservar información sobre un usuario al pasar de una página a otra. Los datos asociados a una sesión se

almacenan en el servidor y nunca en el cliente, aunque PHP necesitará almacenar al menos una cookie en el cliente que contendrá un valor que identifique al usuario en el servidor web (**PHPSESSID**: se guarda automáticamente). En el servidor web estarán almacenados todos los datos de la sesión y se accede a ellos cada vez que se pasa de página gracias a la cookie.

Las variables de sesión están disponibles en todas las peticiones a lo largo del intervalo de tiempo que el usuario las necesite. Son algo así como variables superglobales a la aplicación entera. De forma predeterminada, las variables de sesión duran hasta que el usuario cierra el navegador. El ciclo de vida de una sesión comienza cuando el usuario se conecta a la aplicación y acaba cuando sale de la misma, cierra el navegador o está un tiempo (fijado por el sistema) sin interaccionar con la aplicación.

Todas las variables de sesión se almacenan en el servidor (**\$_SESSION**), por lo tanto, en un momento dado, podemos tener muchas variables con el mismo nombre pero diferentes contenidos. Para saber cual es cual, cada variable de sesión está vinculada a una única sesión/usuario, que se identifica con un identificador de sesión único (PHPSESSID).

Crear una sesión

Si en el fichero php.ini establecemos la directiva session.auto_start=1 se iniciará una sesión automáticamente cada vez que haya un acceso a nuestra aplicación.

Pero lo normal es hacerlo manualmente, usando la función **session_start()**.

Esta función crea una nueva sesión y genera el nuevo identificador, o retorna la sesión en caso de que existiera.

La función session_start() debe llamarse al principio del script (antes de abrir cualquier etiqueta o de imprimir cualquier contenido), sino, dará error.

Las sesiones normalmente se cierran de forma automática cuando PHP termina de ejecutar un script, pero también se pueden cerrar manualmente usando la función **session_write_close()**.

Ejemplo Creación Sesiones

Acceder a las variables de sesión

Accedemos a través de la variable superglobal **\$_SESSION** junto con el índice de la variable. Además, en la página donde hagamos dicho acceso deberemos también iniciar la sesión para comprobar que existe y es la que esperamos que sea.

Ejemplo Lectura de variables de sesión

También se pueden mostrar los valores de una sesión con la instrucción `print_r`: `print_r($_SESSION)`.

Para ver si una variable de sesión ha sido creada se suele utilizar la función **isset()** pasándole como parámetro `$_SESSION` junto con la variable que deseamos saber si ha sido creada.

```
<?php
session_start();
if (isset($_SESSION['usuario'])) {
    echo "Estás identificado como " . $_SESSION['usuario'];
} else {
    echo "No estás identificado";
}
?>
```

ID de una sesión

Cuando se crea una sesión se genera un ID automático de 32 caracteres que se guardará tanto en la sesión como en una cookie del cliente. Así, cada vez que accedemos a una página en la cual nuestra sesión debe estar activa (por ejemplo un usuario identificado), lo que hace PHP es comprobar internamente si el ID de la cookie coincide con el de la sesión.

Dado que todo esto se realiza de forma automática y transparente para el usuario, quizá el desarrollador sí desee acceder en algún momento a dicha información. Esto se consigue mediante la función **session_id()**. Asimismo también podemos acceder fácilmente a dicho valor en la cookie mediante el índice **PHPSESSID**.

Ejemplo de ID de sesión

La cookie se gestiona de forma automática, de manera que no es necesario crearla, renovarla o borrarla ya que todos estos procesos se realizarán de forma automática junto con la sesión correspondiente, incluido su borrado si la sesión es destruida, cosa que sucederá en el momento en que el usuario abandone la web.

No obstante sí puede interesarnos alterar el tiempo de vida de la cookie asociada a la sesión `PHPSESSID` para que, por ejemplo, la sesión caduque con mayor

rapidez, como en el caso de por ejemplo una web bancaria o para que tarde semanas o meses, como puede ser el caso de uncarro de la compra de una tienda online.

Podemos hacer esto mediante la instrucción **ini_set** de la siguiente manera:

```
ini_set('session.cookie_lifetime', 30*60); // 30 minutos
```

Eliminar variables de sesión

Para eliminar una variable de una sesión podemos hacer uso de la función **unset()**

```
<?php
session_start();
unset($_SESSION['usuario']);
?>
```

Para destruir una sesión, es decir, eliminar sus datos así como la cookie asociada a esta con el session_id podemos hacer uso de las instrucciones **session_unset()** y **session_destroy()**:

```
<?php
session_unset();// borra todas las variables de sesión
session_destroy();// destruye la sesión (y su cookie asociada)
?>
```

Podemos saber el estado actual de una sesión con la función **session_status()**, que puede tomar uno de los siguientes valores:

- 0: PHP_SESSION_DISABLED: Las sesiones están deshabilitadas en la configuración de PHP.
- 1: PHP_SESSION_NONE: Las sesiones están habilitadas, pero no se ha iniciado ninguna sesión.
- 2: PHP_SESSION_ACTIVE: Las sesiones están habilitadas y se ha iniciado una sesión.

Hay que tener en cuenta que aunque una sesión ha sido destruida, el estado seguirá siendo PHP_SESSION_ACTIVE hasta que la página se recargue o el usuario acceda a otra página, momento en el cual se volverá PHP_SESSION_NONE hasta que inicies una nueva sesión con session_start(). Esto se debe a que PHP no actualiza el estado de la sesión de inmediato después de destruirla.

[Ejercicio 10 Sesiones](#)

8. Tratamiento de ficheros

Como PHP funciona en un servidor, el tratamiento de archivos se llevará a cabo en este, no en el cliente.

PHP tiene funciones predefinidas para la utilización y manejo de ficheros. Con ellas podremos acceder a un fichero que podamos haber creado previamente, podremos consultar los datos que contenga, añadir otros, modificarlos, o borrarlos.

8.1 Funciones para lectura y escritura de ficheros

Independientemente de lo que hagamos con un fichero, su tratamiento en PHP se traduce en la realización de una secuencia de pasos:

1. Apertura del fichero: **fopen()**
2. Manipulación del fichero (lectura o escritura): **fread()**, **fwrite()**
3. Cierre del fichero: **fclose()**

8.1.1 Apertura de un fichero

fopen (nombre, modoApertura [,ruta de búsqueda])

Si el fichero no se pudiera abrir, fopen() devolverá FALSE.

Parámetros:

- nombre: hace referencia a un archivo, que puede estar en local o remoto (URL). El fichero debe ser accesible para PHP, por lo que es necesario asegurarse de que los permisos de acceso del fichero permiten este acceso.
Si al fichero se accede a través de una URL, PHP revisará que **allow_url_fopen** está habilitado. Si es desactivado, PHP emitirá un aviso y la llamada a fopen fallará.
- modoApertura:
 - 'r' : Sólo lectura; coloca el puntero al fichero al principio del fichero.
Indicamos 'rb' para que la apertura sea en modo binario y evitar problemas. Esto significa que el archivo se leerá en su forma binaria original y no se realizará ninguna interpretación especial de los datos.
 - 'r+' : Lectura y escritura; coloca el puntero al fichero al principio del fichero.

- 'w' : Sólo escritura; coloca el puntero al fichero al principio del fichero. Si el fichero no existe se intenta crear, si ya existe, borra su contenido..
- 'w+' : Lectura y escritura; coloca el puntero al fichero al principio del fichero.
- 'a' : Modo adición: Sólo escritura; coloca el puntero del fichero al final del mismo. Si el fichero no existe, se intenta crear.
- 'a+' : Lectura y escritura; coloca el puntero del fichero al final del mismo. Si el fichero no existe, se intenta crear.
- 'x' : Creación y apertura para sólo escritura; coloca el puntero del fichero al principio del mismo. Si el fichero ya existe, la llamada a fopen() fallará devolviendo FALSE y generando un error. Si el fichero no existe se intenta crear.

En términos generales usaremos los modos r para sólo leer, w para sólo escribir, w+ para actualizar y a+ para añadir al final del archivo con posibilidad de leer.

`$gestor = fopen("/test/fichero.txt", "rb");` //abre fichero.txt del directorio test para solo lectura y en modo binario

`$gestor = fopen("/test/fichero2.txt", "w");` //abre fichero.txt del directorio test para escritura; si el fichero no existe se intenta crear

`$gestor = fopen("http://www.audiogil.com/texto.txt", "r");` //abre el fichero texto.txt ubicado en una URL para su lectura

- ruta de búsqueda: si PHP no encuentra el fichero en la ruta especificada junto con el nombre, y el parámetro ruta de búsqueda está a 1, entonces PHP lo buscaría también en los directorios definidos en la directiva include_path del fichero de configuración php.ini.

8.1.2 Cierre de un fichero

fclose (descriptor_fichero)

Devuelve TRUE si el archivo se ha cerrado correctamente.

El parámetro descriptor_fichero debe ser un puntero a un fichero abierto.

Es conveniente cerrar todos los ficheros que ya no se vayan a usar, de todos modos, PHP los cerrará automáticamente al terminar de ejecutar el script.

`$gestor = fopen('archivo.txt', 'r');`

```
fclose($gestor); // Cierra el archivo
```

8.1.3 Lectura de un fichero

fread (descriptor_fichero[,caracteres a leer])

Devuelve la cadena del fichero con la cantidad de caracteres indicados (si se han indicado) o hasta el fin de fichero.

```
$nombre_fichero = "/usr/local/algo.txt";
$gestor = fopen($nombre_fichero, "rb");
$contenido = fread($gestor, filesize($nombre_fichero));
fclose($gestor);
```

Hay que tener en cuenta cuando se lee desde algo que no es un fichero local, que la lectura se detendrá después de que esté disponible un paquete. Esto significa que hay que reunir la información en trozos.

```
$gestor = fopen("http://www.example.com/", "rb");
$contenido = "";
while (!feof($gestor)) {
    $contenido .= fread($gestor, 8192); //se lee en bloques de 8192 bytes (8Kb)
}
fclose($gestor);
```

Si no se especifica un tamaño de bloque en la función `fread`, por defecto se intentará leer todo el contenido del recurso en una sola operación. Esto significa que se tratará de leer todo el contenido de la URL de golpe y se almacenará en la variable `$contenido` en una sola vez. Si el contenido es muy grande, esto puede consumir una cantidad significativa de memoria, lo que podría ser ineficiente o incluso provocar problemas de rendimiento en el servidor.

Hay otras formas de leer archivos:

file_get_contents ('fichero')

Se utiliza para leer todo el contenido de un archivo o una URL en una sola llamada y lo devuelve como una cadena.

Es una función más simple y conveniente para leer archivos completos o contenido web en una sola operación.

No es necesario abrir ni cerrar manualmente el recurso, ya que la función maneja eso automáticamente.

```
$contenido = file_get_contents('datos.txt');
echo $contenido;
```

file ('fichero')

Lee todo el contenido del fichero y lo devuelve en forma de array: una línea en cada posición. Cada item en la matriz corresponde a una línea en el archivo.

Carga todo el contenido del archivo o de la URL en memoria a la vez, por lo que no es adecuada para manejar archivos o recursos grandes, ya que podría agotar la memoria del servidor.

Esta función necesita el nombre del fichero y no el descriptor (puntero)

```
$lineas = file('datos.txt');
foreach ($lineas as $numero => $linea) {
    $numero_de_linea = $numero + 1;
    echo "Línea $numero_de_linea: $linea";
}
```

El resultado es:

Línea 1: Contenido primera línea del archivo

Línea 2: Contenido segunda línea del archivo

Línea 3: Contenido tercera línea del archivo

fgets (descriptor_fichero)

Lee una única línea del archivo y la devuelve como una cadena. Es útil para procesar un archivo línea por línea, ya que lee una línea a la vez y avanza automáticamente al siguiente punto de lectura.

```
$gestor = fopen("archivo.txt", "r"); // Abre el archivo en modo lectura ("r")
if ($gestor) {
    while (!feof($gestor)) {
        $linea = fgets($gestor); // Lee una línea completa del archivo
        echo $linea; // Hacer algo con la línea (por ejemplo, imprimirla)
    }
    fclose($gestor); // Cierra el archivo
}
```

Ejemplo lectura de fichero

fgetcsv (descriptor_fichero [, longitud máxima de línea] [, 'separador'])

Se utiliza para leer una línea de un archivo CSV y convertirla en un array. Cada valor separado por comas (si no se indica lo contrario) en la línea se almacena como un elemento en el array.

Si el archivo CSV utiliza un separador diferente de la coma (,), como el punto y coma (;) o un tabulador (\t), se especifica ese separador como el segundo argumento de la función.

```
$archivo = fopen("datos.csv", "r"); // Abre el archivo CSV en modo lectura ("r")
while (($fila = fgetcsv($archivo, 0, ";")) !== false) {
    // $fila es un array que contiene los valores de la línea actual
    print_r($fila); // Imprime la fila como un array
}
fclose($archivo); // Cierra el archivo
```

Ejemplo lectura de fichero CSV

8.1.4 Recorrer un fichero

Funciones de PHP para movernos a determinados puntos de un fichero:

rewind (descriptor)

Rebobina la posición del puntero, es decir, sitúa el puntero de lectura/escritura al principio del fichero.

fseek (descriptor, bytes de desplazamiento [, desde donde deslaza])

Desplaza la posición del puntero la cantidad de posiciones indicada.

El tercer parámetro puede ser:

- SEEK_SET : desplazamientos desde el principio
- SEEK_CUR : desplazamientos desde la posición actual
- SEEK_END: desplazamientos desde el final (entonces serán negativos)

ftell (descriptor)

Obtiene la posición actual del puntero

feof (descriptor)

Indica si el puntero está al final del archivo

fsize (descriptor)

Devuelve la longitud de un fichero (bytes)

8.1.5 Escritura en un fichero

fwrite (descriptor, cadena [, número de caracteres])

Permite escribir en un archivo en modo binario seguro.

Devuelve el número de bytes escritos, o FALSE si se produjo un error.

[Ejemplo escritura en fichero modo w](#)

[Ejemplo escritura en fichero modo a](#)

8.1.6 Borrado de un fichero

unlink (nombre_fichero)

Elimina el fichero y devuelve TRUE en caso de éxito o FALSE en caso de error.

[Ejercicio 11 Ficheros](#)

9. Programación orientada a objetos con PHP

PHP es un lenguaje orientado a objetos. LA POO (Programación Orientada a Objetos) implica abstraer el problema que debemos abordar de manera que podamos aislar de forma atómica los protagonistas de este en forma de objetos. Estos tendrán unas propiedades que los definan y sobre ellos podremos realizar determinado tipo de operaciones conocidas como métodos.

La POO proporciona, además, funcionalidades adicionales como la encapsulación (acceso restringido a la definición de una clase), herencia (la posibilidad de desarrollar clases y objetos a partir de otros ya existentes) y el polimorfismo (la posibilidad de dotar de adaptabilidad a nuestros objetos pudiendo comportarse de una forma u otra frente a distintas situaciones).

9.1 Clases, Métodos y propiedades

Un objeto es como un tipo de dato especial y complejo. Podemos decir que una clase es un tipo de dato que contiene a modo de estructura un conjunto de variables y funciones asociadas todas ellas a ese objeto en cuestión. Una clase nos permitiría crear tantos objetos como necesitemos en nuestra aplicación, lo que se conoce como instanciar.

Conviene poner los nombres de las clases siempre en singular y la primera letra de cada palabra en mayúsculas (CamelCase), al revés de los nombres de las tablas de una base de datos, generalmente en plural y en minúsculas:

Tablas en base de datos: *personas, animales, usuarios, usuarios_administradores*

Clases en POO: *Persona, Animal, Usuario, UsuarioAdministrador*

```
Class PaginaWeb {
    $titulo;
    Function setTitulo ($titulo = "Titulo por defecto") {
        $this->titulo = $titulo; // $this es obligatorio aunque estemos en la clase
    }
}
```

9.1.1 Instancia de clase

Instanciar un objeto es crear un objeto perteneciente a esa clase. Se utiliza **new**.

```
$pagina = new PaginaWeb();
```

Acceder a los métodos y propiedades:

```
$pagina -> setTitulo("Mi nueva web");
```

Aunque de esta manera ya podríamos trabajar sin problemas con este objeto, lo ideal es definir, además, un método constructor.

9.1.2 Método constructor

Método especial que se desarrolla para inicializar todas las cosas que necesitamos para empezar a usar un objeto en condiciones y evitar así tener que hacer uso de llamadas a otros métodos. PHP, al igual que la mayoría de los demás lenguajes con soporte para objetos proporciona un constructor por defecto cuando se instancia a un objeto mediante la instrucción `new`, pero siempre tenemos la posibilidad de crear el nuestro, que sustituiría al que incorpora el lenguaje.

Los constructores se crean como una función con el nombre: **__construct()**.

```
function __construct ($titulo) {
    $this->setTitulo ($titulo);
}
```

En el ejemplo el constructor se encargará, mediante el paso del parámetro `$titulo`, de insertar directamente el título de la página cuando creamos el objeto, por lo que nos ahorramos tener que invocar al método `setTitulo`. De esta manera, la creación del objeto quedaría así:

```
$pagina = new PaginaWeb ("Mi nueva web");
```

9.1.3 Método destructor

Un destructor es un método que se encargará de realizar las operaciones oportunas justo antes de que el objeto sea destruido. El método destructor será llamado tan pronto como no haya otras referencias a un objeto determinado o en cualquier otra circunstancia de finalización.

Para crear un destructor se utiliza el método **__destruct()**.

```
function __destruct() {
    unset($this->titulo); //eliminamos la propiedad $titulo
}
```

9.2 Herencia

10. PHP y las Bases de Datos

Para conectar a la BBDD:

- a través de unas [extensiones](#) específicas (api MySQL): obsoleto
- funciones MySQLi (pag 103) para lenguaje normal (PHP)
- funciones MySQLi (pag) para lenguaje orientado a objetos (PHP): utiliza el operador flecha
- **utilizando una interfaz ([PDO](#))**, que independientemente de la bbdd que se use, implementa unas funciones para manejarla (sólo a partir de PHP 5.1, y requiere una extensión)
- a través de la extensión [ODBC](#): ya no se suele usar.

Bibliografía

Comesaña, José Luis. **Desarrollo Web Entorno Servidor**.

<https://www.sitiolibre.com/curso/pdf/DWES01.pdf>

López Sanz, Marcos et. al. **Desarrollo Web en Entorno Servidor**. Editorial Ra-Ma.

Palomo Duarte, Manuel, Montero Pérez, Ildefonso. **Programación en PHP a través de ejemplos**. Apuntes de la asignatura “Programación para Internet”, Ingeniería Técnica en Informática de Gestión.

Sánchez, José Luís. **Introducción al lenguaje PHP**. Documento incluido en el classroom de la asignatura.

Sintes Marco, Bartolomé. **Programación web en PHP**.

<https://www.mclibre.org/consultar/php/>

Universidad Nacional de Educación a Distancia. **Desarrollo de aplicaciones web dinámicas. Volumen 4**. Fundación General de la UNED.