



UD1. Desarrollo Web en Entorno Servidor

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

Curso 2024-2025

Contenidos

- Arquitecturas cliente/servidor.
- Aplicaciones Web.
- Lenguajes de programación en entorno servidor.
- Herramientas de programación. IDE
- Tecnologías de Servidor

Modelos de programación en entornos cliente/servidor

- URL (Uniform Resource Locator)

Protocolo://nombrededominio/recurso

Protocolo://maquina[:puerto]/recurso

Ejemplo:

<https://www.microsoft.com/es-es/windows/>

<https://23.214.202.161/windows>

Actividad:

Acceder a la carpeta docs de la página de PHP con cada uno de los formatos de arriba

Arquitectura Cliente-Servidor

La arquitectura cliente/servidor está basada en la idea de **servicios**:

- El cliente solicita servicios (**REQUEST**)
- El servidor es un proceso proveedor de dichos servicios (**RESPONSE**)

La comunicación entre ambos se realiza mediante el intercambio de mensajes.

El cliente, a través de un navegador, inicia el intercambio de información, solicitando datos al servidor.

El servidor responde enviando uno o más flujos de datos al cliente.

Arquitectura Cliente-Servidor

La arquitectura Cliente-Servidor contempla varias capas:

- **Presentación:** Esta capa se comunica únicamente con la capa de negocio
- **Lógica o de Negocio:** Es la capa intermedia, Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, y con la capa de persistencia, para solicitar al gestor de base de datos almacenar o recuperar datos de él.
- **Persistencia:** Es la capa donde se encuentran los datos almacenados para generar información

Capa Presentación

- Es la capa que interactúa con el usuario
- Conocida como interfaz gráfica de usuario (IGU)
- Debe ser «amigable» para el usuario
- Esta capa se comunica únicamente con la capa de negocio

Capa Lógica o Negocio

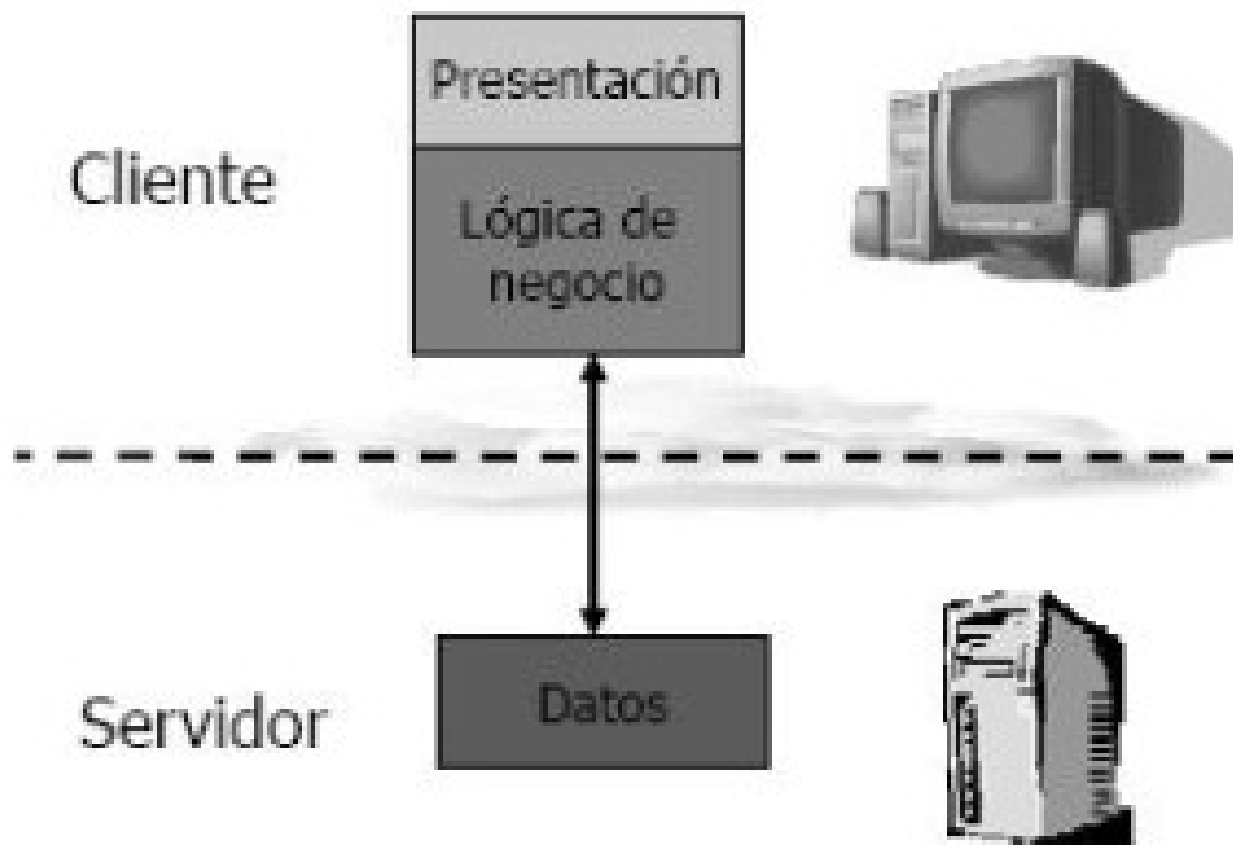
- Es donde residen los programas que se ejecutan.
- Recibe las peticiones del usuario y se envían las respuestas tras el proceso
- Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse.

Capa Persistencia o Datos

- Donde residen los datos.
- Encargada de acceder a los mismos.
- Formada por uno o más SGBD que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

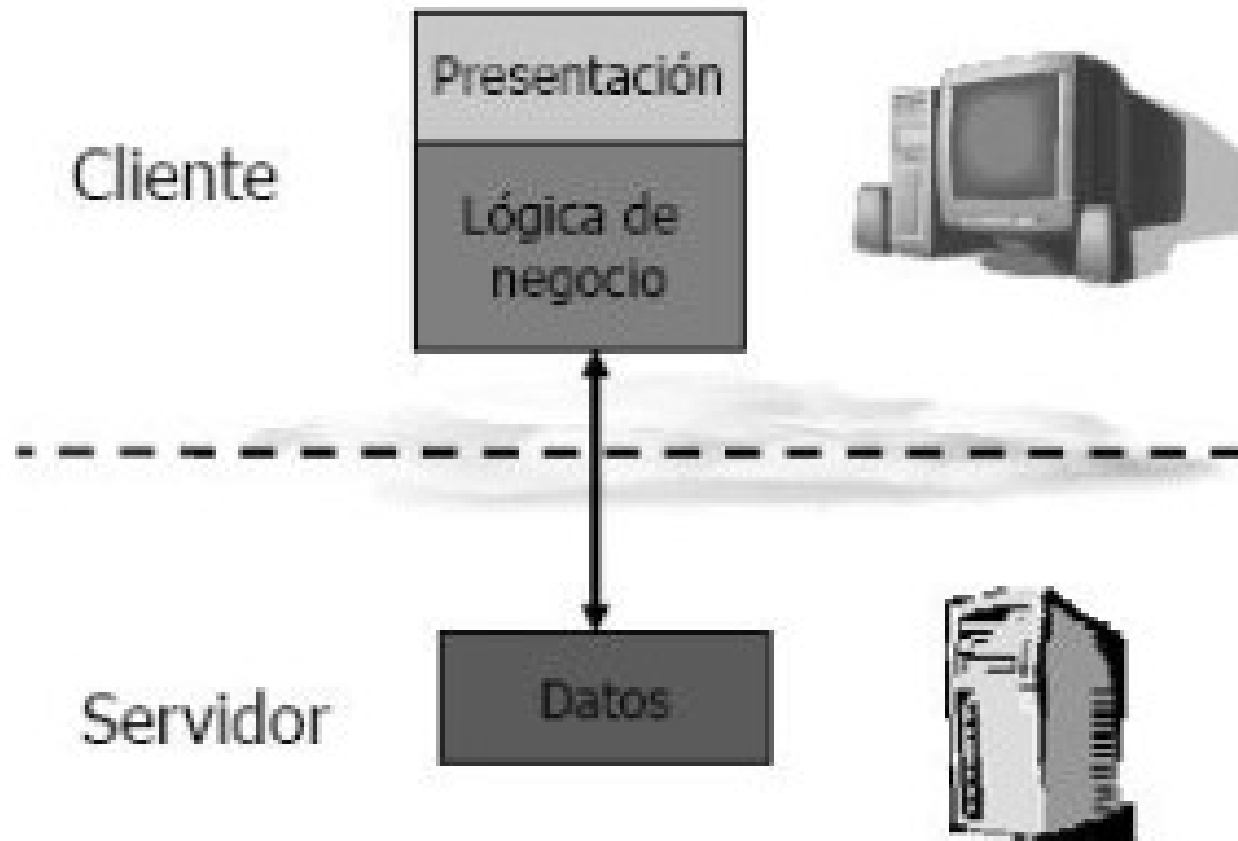
Cliente-Servidor de 2 capas

El servidor responde con sus propios recursos, no requiere otra aplicación/servidor para proporcionar parte del servicio



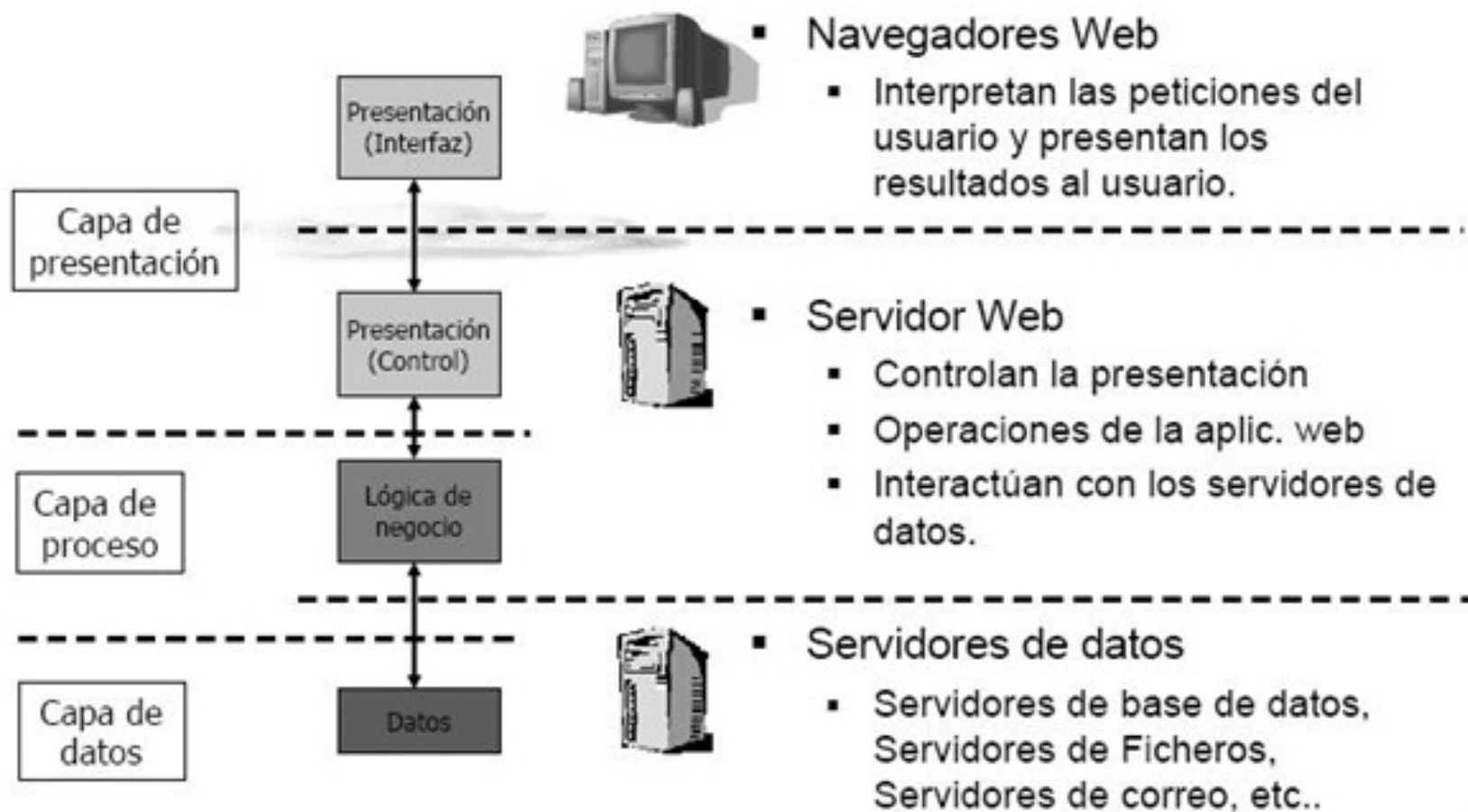
Cliente-Servidor de 2 capas

Actividad: Piensa un ejemplo de esta arquitectura. Comparte con el grupo de forma razonada tu ejemplo



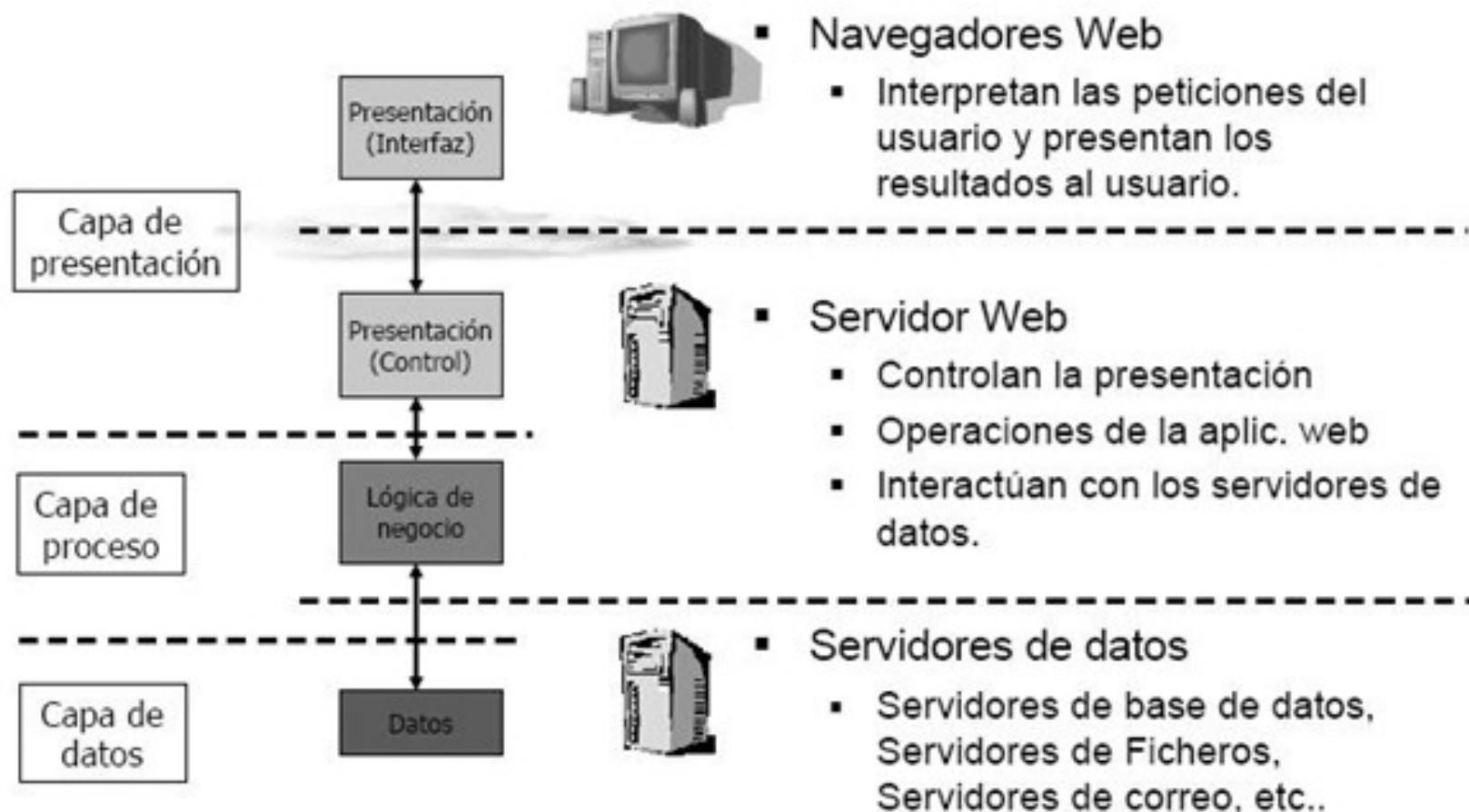
Cliente-Servidor de 3 capas

El servidor requiere otra aplicación/servidor para proporcionar parte del servicio



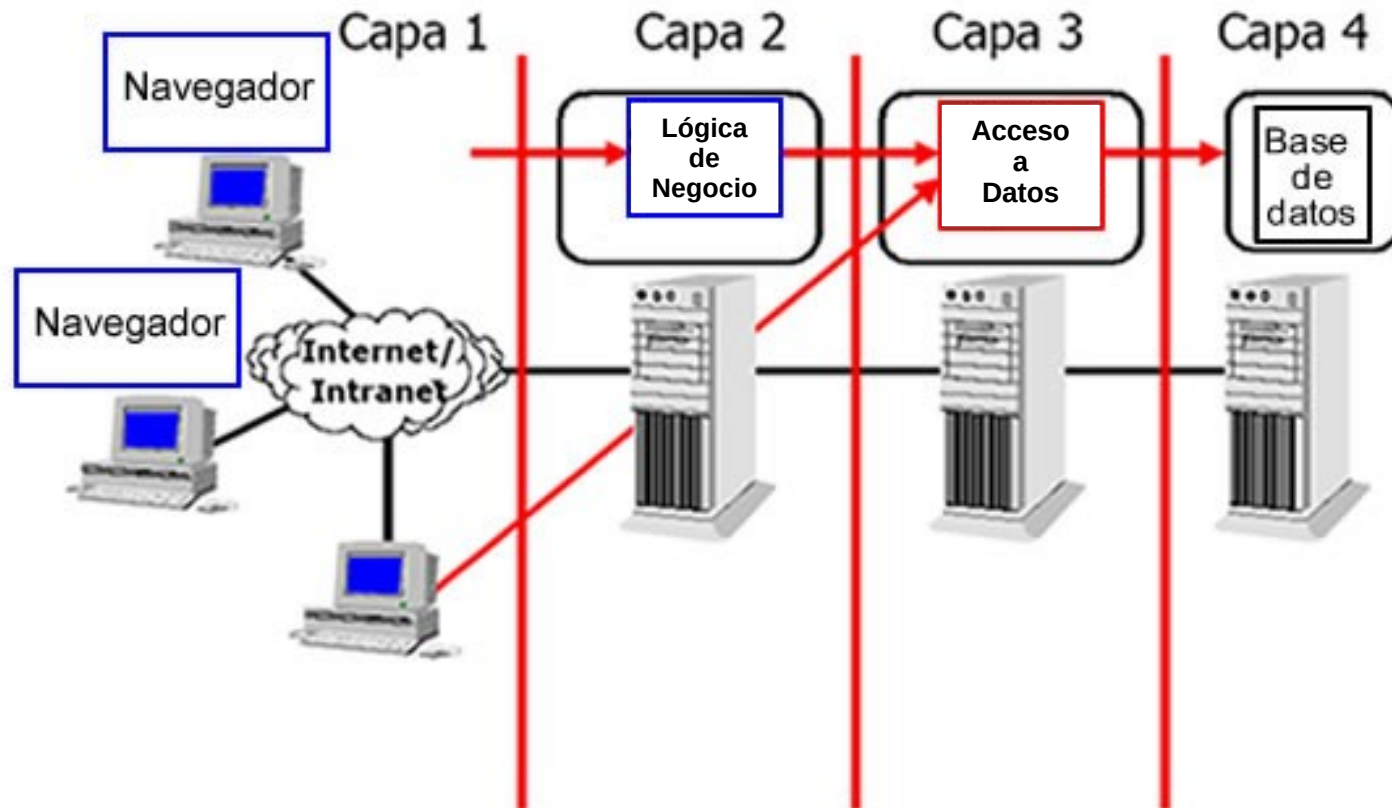
Cliente-Servidor de 3 capas

Actividad: Piensa un ejemplo de esta arquitectura. Comparte tu ejemplo razonadamente



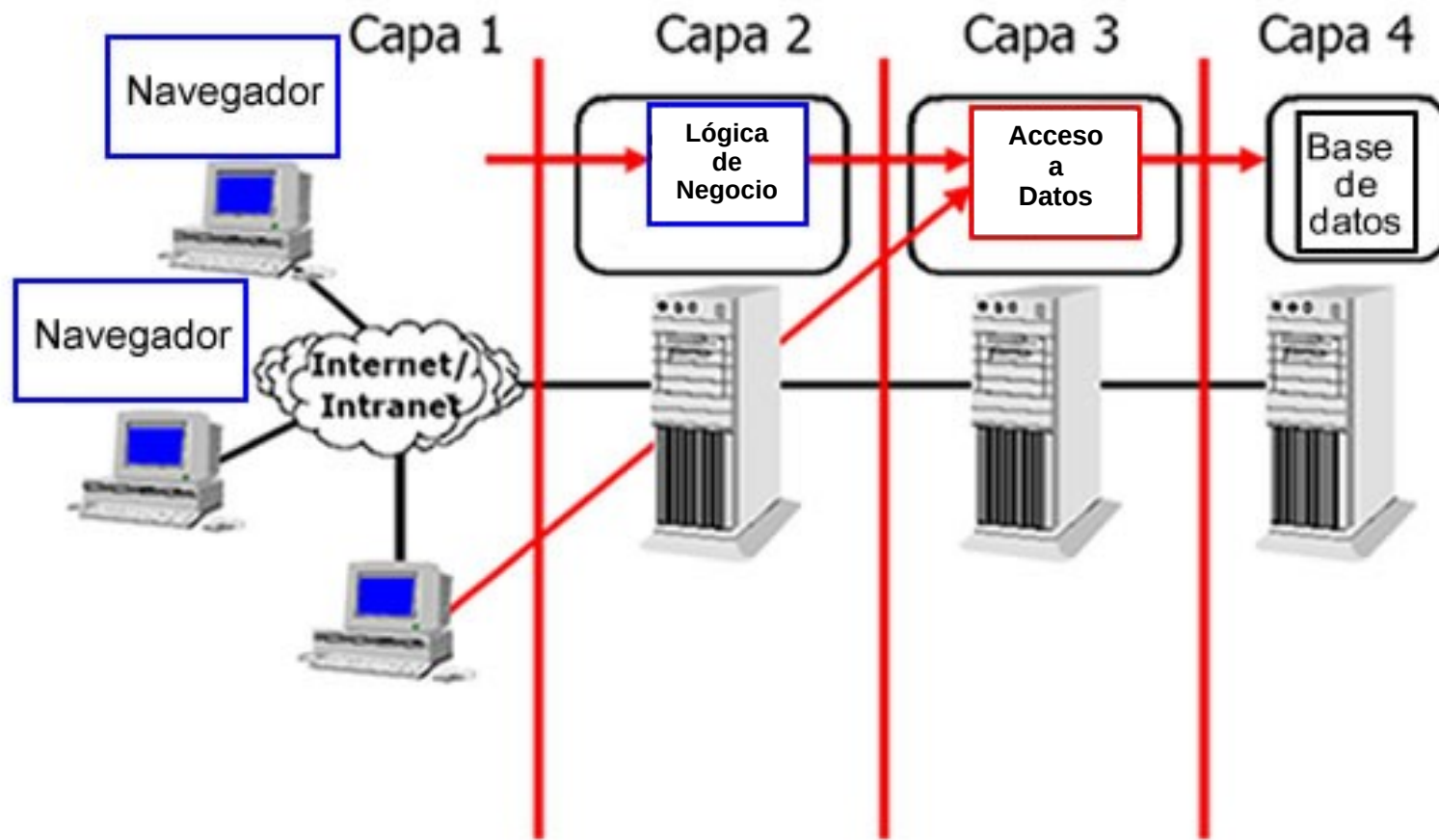
Cliente-Servidor de N capas

Los diferentes procesos están distribuidos en diversas capas, lógicas y/o físicas, ejecutándose en diferentes equipos



Cliente-Servidor de N capas

Actividad: Piensa un ejemplo de esta arquitectura. Comparte tu ejemplo razonadamente



Aplicaciones Web

- Proporcionada por un servidor Web y utilizada por usuarios que se conectan desde cualquier punto vía clientes web (navegadores)
- La colección de páginas son en una buena parte dinámicas (ASP, PHP, etc.), y están agrupadas lógicamente para dar un servicio al usuario.
- El acceso a las páginas está agrupado también en el tiempo (sesión).
- Ejemplos: venta de libros, reserva de billetes, etc.

Aplicaciones Web

Al observar la capacidad de las aplicaciones web de comunicarse con los usuarios, las podemos clasificar en:

- Aplicaciones Web Estáticas
- Aplicaciones Web Dinámicas
- Aplicaciones Web Interactivas

Aplicaciones Web Estáticas

El cliente recibe una página web desde el servidor, que no conlleva ningún tipo de acción, ni en la propia página, ni genera ninguna respuesta por parte del servidor. Utilizan HTML para la organización visual de la información.

```
001 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
002 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es-ES" lang="es-ES">
003     <head>
004         <title>Título de la página</title>
005         <meta http-equiv="content-type" content="text/html; charset=utf-8">
006     </head>
007     <body>
008         <p>Ha accedido a las 9:20</p>
009     </body>
010 </html>
```

Aplicaciones Web Dinámicas

La interacción del cliente con la página web recibida desde el servidor produce algún cambio en la visualización de la misma (cambio de formato, ocultación de partes de la página, comienzo de animaciones, aparición de elementos nuevos, etc.). Incluyen DHTML, Flash, CSS, JavaScript, etc.

```
001 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
002 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es-ES" lang="es-ES">
003     <head>
004         <title>Título de la página</title>
005         <meta http-equiv="content-type" content="text/html; charset=utf-8">
006     </head>
007     <body>
008         <p>Ha accedido a las <script type='text/javascript'>var hora=new
Date ();document.write(hora.getHours()+':'+hora.getMinutes());</script></p>
009     </body>
010 </html>
```

Aplicaciones Web Interactivas

- La interacción del cliente con la página web recibida desde el servidor hace que se genere un diálogo entre ambos.
- Son las aplicaciones que más se utilizan en Internet actualmente.
- En el lado cliente encontramos HTML, controles ActiveX, Flash, applets, AJAX, etc.
- En el lado servidor se utilizan lenguajes embebidos en código HTML como PHP, ASP, JSP, enlaces a ejecutables CGI, servlets, ASP.net.

Ejecución de código en un servidor Web

Para ejecutar código en un servidor Web se requieren los siguientes componentes:

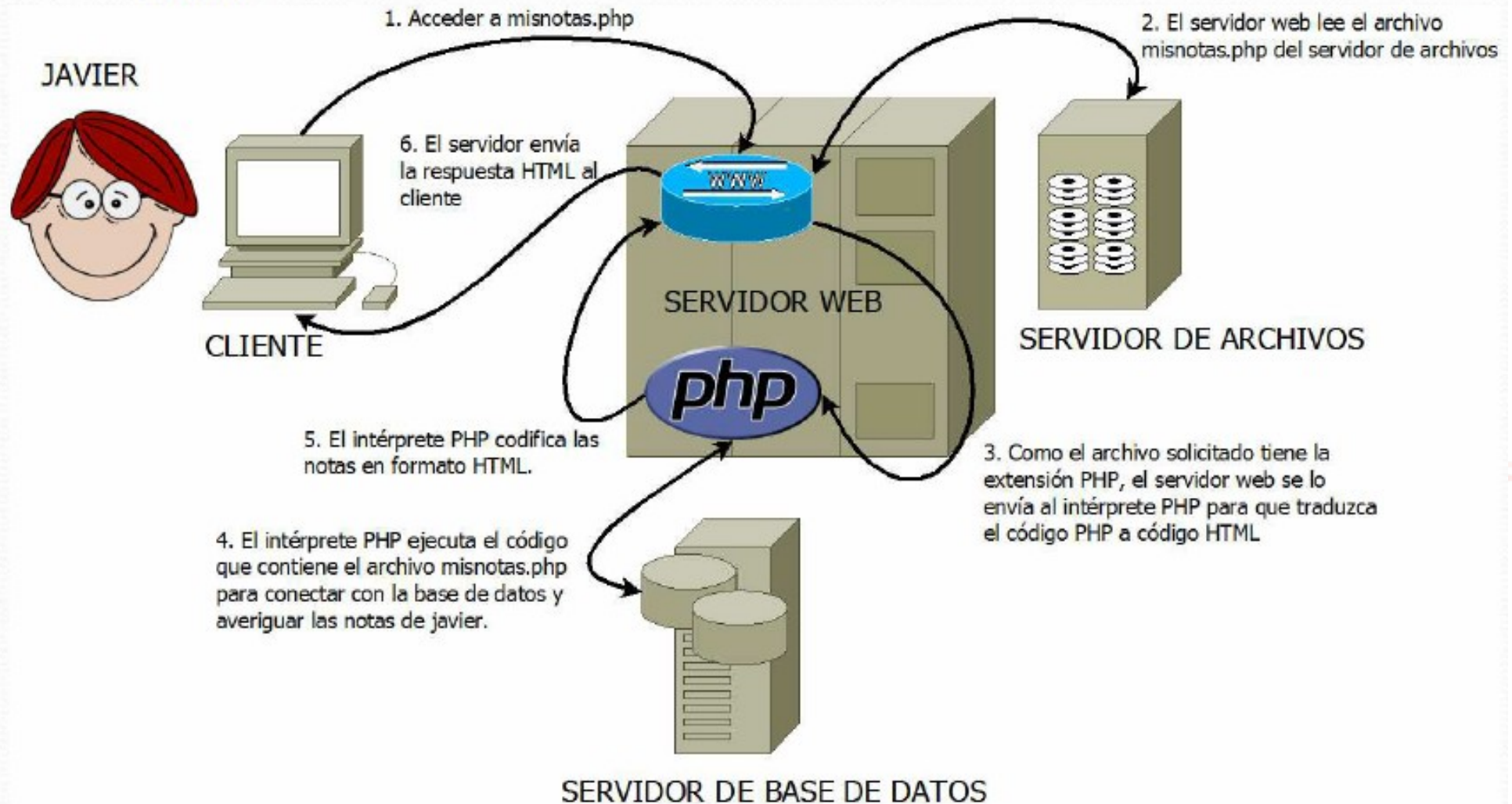
- **Servidor Web:** recibe las peticiones, elige el módulo que se encarga de la ejecución del código y se comunica con él
- **Módulo encargado de ejecutar el código:** para poder generar la página web resultante debe integrarse con el servidor web y depende del lenguaje y tecnología en que se programa la aplicación web
- **Aplicación de BD:** normalmente será un SGBD que almacene los datos
- **Lenguaje de programación** en el que se desarrollan las aplicaciones

Servidores de Aplicaciones

- Los **servidores de aplicaciones** ejecutan aplicaciones web o aplicaciones de escritorio.
- Constan lenguajes de programación, conectores de BD y el código necesario para implementar, configurar, administrar y conectar estos componentes en un servidor Web
- Típicamente incluyen también middleware (software de conectividad) que les permite intercomunicarse con varios servicios, para confiabilidad, seguridad, no-repudio, etc.

Proceso consulta Aplicación Web

El proceso que se realiza a la hora de visitar una página PHP sería:



Lenguajes de programación

Los lenguajes de entorno servidor pueden ser:

- Lenguajes de scripting o interpretados
- Lenguajes compilados a código nativo
- Lenguajes compilados a código intermedio

Lenguajes de Scripting o interpretados

Los programas se ejecutan directamente a partir de su código fuente mediante un intérprete que procesa las líneas del programa y genera como resultado el código HTML

No se compilan, se interpretan las instrucciones de script directamente por lo que sólo requieren el intérprete

A este grupo pertenecen los lenguajes Perl, Python, PHP, Ruby y ASP

Lenguajes compilados a código nativo

El código fuente se traduce a código binario (dependiente del procesador) antes de ser ejecutado. De este modo, el programa en binario se ejecuta directamente al invocarlo desde el servidor Web

A este grupo pertenece los CGI (Common Gateway Interface) que se programan en lenguajes que produzca un ejecutable como C, Perl, C++, Java, etc.

Lenguajes compilados a código intermedio

El código fuente original se traduce a un código intermedio (bytecode, independiente del procesador) antes de ser ejecutado en varias plataformas distintas

A este grupo pertenecen los lenguajes Java Server Pages o JSP que se ejecuta en servidores que permiten Servlets (código embebido que es compilado y almacenado en el contenedor de servlets del servidor y, tras la primera ejecución, se carga en memoria para agilizar posteriores invocaciones. El servlet queda activo escuchando peticiones para servirlos)

Patrón de diseño MVC

En el desarrollo Web se usa el patrón de diseño **MVC** (Model, View, Controller) para separar las capas de persistencia o datos (modelo), presentación o vista y lógica o de negocio (controlador). Así pues:

- El **Modelo** contiene una representación de los datos que maneja el sistema y sus mecanismos de persistencia.
- La **Vista**, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos interacción con éste.
- El **Controlador** actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de la lógica o modelo de negocio de cada aplicación.

Herramientas de programación

Los instrumentos involucrados en el desarrollo web se pueden clasificar en:

- Navegadores: Internet Explorer, Google Chrome, Mozilla firefox, etc.
- Editores de documentos
- Entornos de programación (IDE): Eclipse, NetBeans, Visual Studio Code, Dreamweaver, FrontPage, Sublime Text, etc.
- Herramientas para la creación y administración de bases de datos
- Herramientas para el tratamiento de imágenes.

Entornos de Desarrollo Integrado

Los Entornos de Desarrollo Integrados (IDE) aportan los siguientes elementos:

- Resaltado de texto: Distinto color para los diferentes elementos del lenguaje: sentencias, variables, comentarios, etc.
- Indentado automático para diferenciar de forma clara los distintos bloques de un programa.
- Completado automático.
- Navegación en el código. Permite buscar de forma sencilla elementos dentro del texto

Entornos de Desarrollo Integrado II

- Generación automática de código creando la estructura básica (esqueleto o stub), para que sólo haya que rellenarla.
- Ejecución y depuración.
- Gestión de versiones. Combinado con un sistema de control de versiones, el IDE te puede ayudar a guardar copias del estado del proyecto a lo largo del tiempo, para poder revertir los cambios realizados.

Frameworks

Un Framework es una estructura o esqueleto software que permite aplicar unos patrones de diseño (en nuestro caso MVC) en la construcción de nuestra aplicación Web.

Entre muchas ventajas, permite la modularidad y la reusabilidad de código permitiendo aplicar múltiples interfaces para distintas vistas (web, móvil, servicio web, etc.)

Algunas funcionalidades que aporta son:

- Sistema de plantillas Web
- Mecanismos de seguridad y autenticación
- Acceso, mapeo y configuración de BD
- Mapeo de “URL amigable” (Ej: `/product.php?cat=home&topic=chair` puede accederse desde `/product/home/chair`)

Tecnologías de Servidor

Según el modo de procesar las peticiones del cliente, podemos clasificar los servidores web en:

- Basados en Procesos
- Basados en Hilos
- Dirigidos por Eventos
- Implementados en el Kernel

Servidor basado en Procesos

1. Un proceso principal escucha posibles peticiones de clientes
2. Cuando llega una petición, el proceso se duplica creando una copia exacta (*fork*)
3. La copia atiende la petición mientras el proceso principal sigue escuchando nuevas peticiones

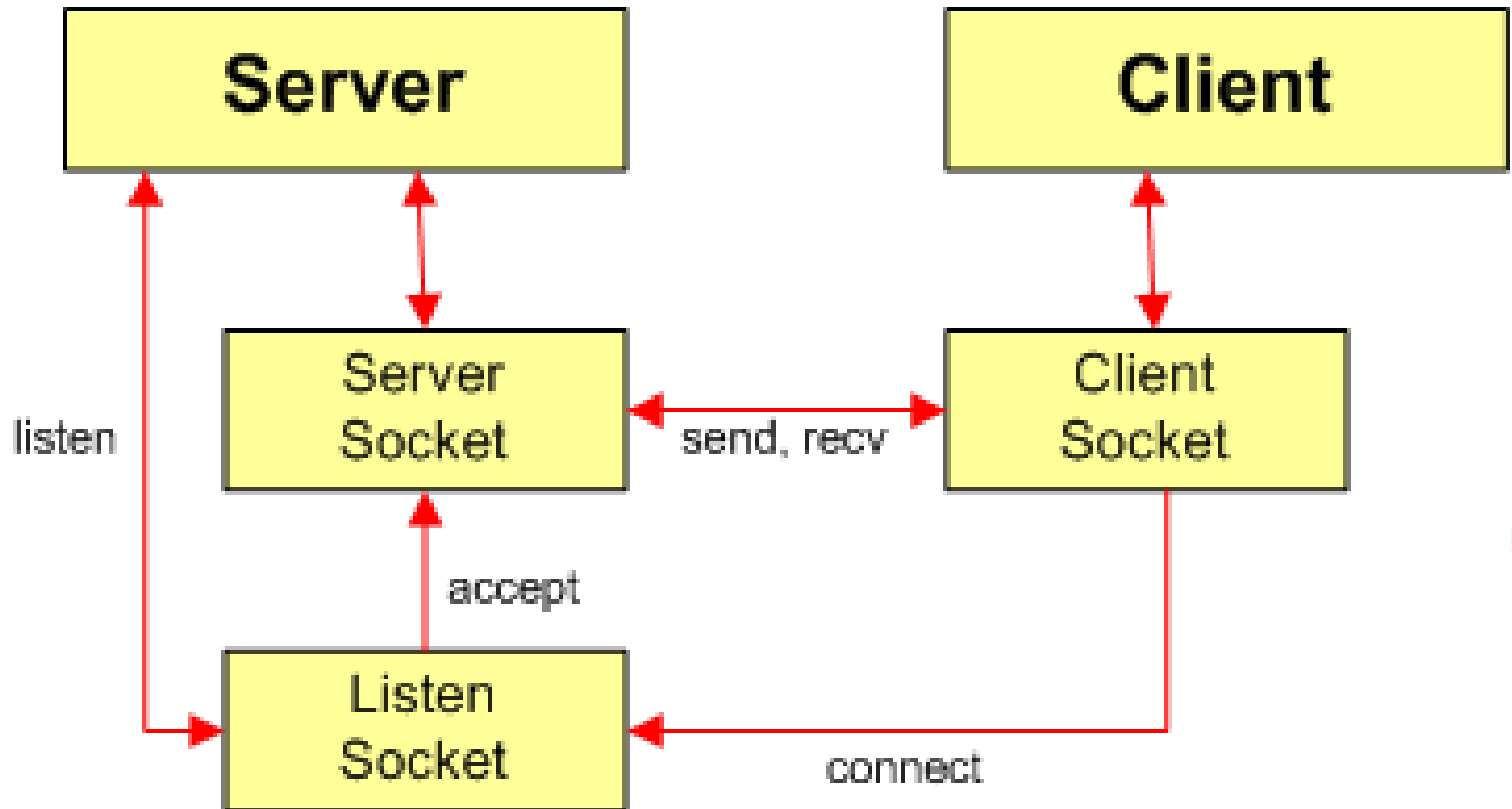
Servidor basado en Hilos

1. Un proceso principal escucha posibles peticiones de clientes
2. Cuando llega una petición, el proceso crea un Hilo de ejecución (*thread*) que comparten el mismo espacio de memoria (interbloqueo)
3. El hilo atiende la petición mientras el proceso principal sigue escuchando nuevas peticiones

Servidor dirigido por eventos

- Se basa en *sockets* no bloqueantes.
- Socket: espacio de memoria para compartir entre procesos de dos aplicaciones en dos máquinas distintas (cliente/servidor)
- Las lecturas/escrituras entre sockets son asíncronas y bidireccionales
- Se identifican mediante IP:puerto
- La concurrencia de procesamiento es simulada: hay un único proceso y un sólo hilo atendiendo las conexiones gestionadas por sockets

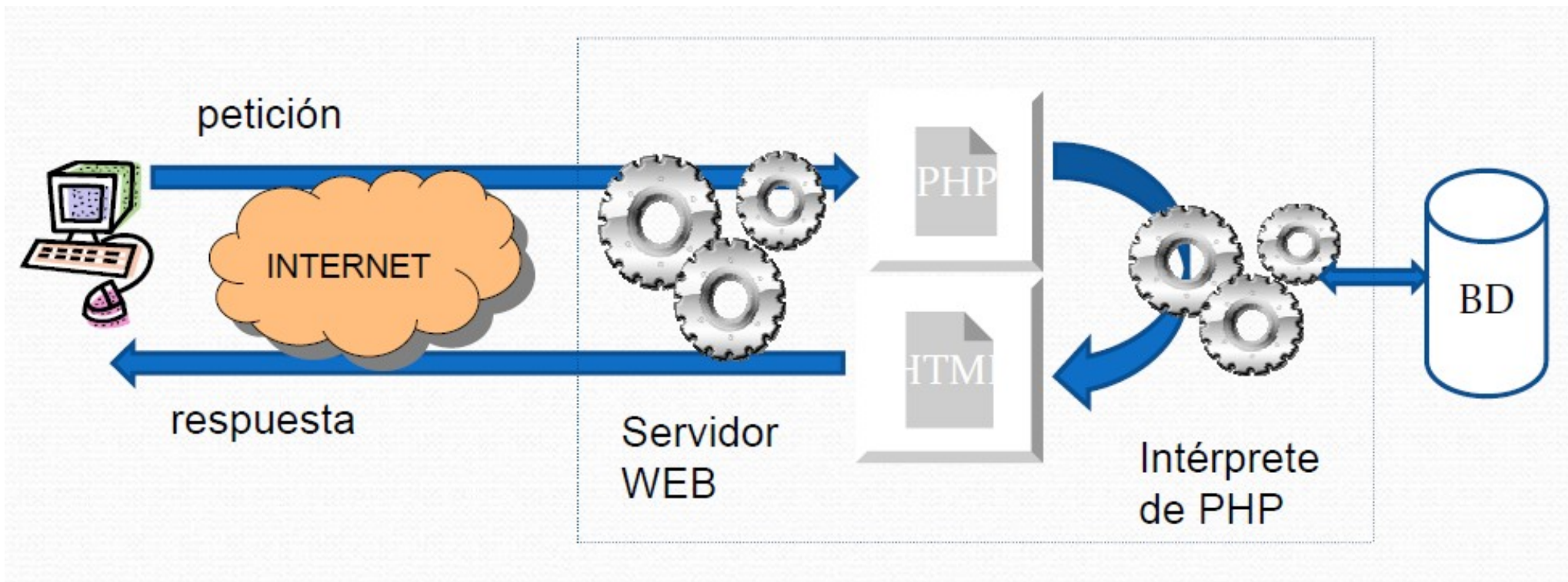
Servidor dirigido por eventos



Servidor implementado en el Kernel

- Se usa un espacio de trabajo perteneciente al SO y no en el área de usuario
- En la práctica o mundo real, tiene muchos problemas e inconvenientes
- Cualquier problema que se produce a nivel de Kernel puede inutilizar el SO

Obtención del lenguaje de marcas para mostrar en el cliente



El documento PHP, una vez interpretado correctamente en el servidor, produce una página HTML que será enviada al cliente.

Tarea Servidores

De los servidores Apache, Internet Information Services, Nginx, Lighttpd y Sun Java System Web Server deberéis investigar:

- Plataforma en la que se ejecutan
- Lenguajes de programación que interpretan
- Propietario
- Cuota de Mercado de 2023 (o 2024 si está disponible)

Realizad una tabla comparativa tras el estudio con una conclusión personal y razonada eligiendo uno de ellos.

Subid el documento con el nombre

Alumno_Servidores.odt como por ejemplo

SilviaVilar_Servidores.odt



UD2. Introducción a PHP

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Características PHP.
- Etiquetas para inserción de código.
- Uso de directivas. PHP.ini
- Ámbito de las variables.
- Constantes.
- Variables Superglobales
- Variables variables
- Tipos de datos. Conversiones entre tipos de datos.
- Sintaxis básica de PHP

Características PHP

- El código PHP está embebido en documentos HTML, para introducir dinamismo fácilmente a un sitio web.
- El intérprete PHP ignora el texto del fichero HTML hasta que encuentra una etiqueta de inicio del bloque de código PHP embebido.
- Como PHP se ejecuta del lado del servidor sólo puede tener acceso a los datos del propio servidor.
 - ✓ No puede acceder a los recursos del cliente
 - ✓ No puede saber qué hora es en el cliente
 - ✓ No puede acceder a los archivos del cliente
(Con la excepción de las Cookies)

Etiquetas para inserción de código

Hay varias formas para delimitar los bloques de código PHP:

1. La opción que asegura portabilidad (**recomendada**):

<?php *codigo php* **?>**

2. Usando las short tags o etiquetas de formato corto:

Requiere activar directiva short_open_tag=on

<? código php ?>

3. Podemos asociar el lenguaje a etiquetas de scripting (**Eliminada** desde la versión 7.0):

<script language="PHP"> *codigo php* **</script>**

4. Usando las etiquetas de ASP (**Eliminada** desde la versión 7.0): **Requiere activar directiva asp_tags=on**

<% código php %>

NOTA: el delimitador **<?=>** es abreviatura de **<?php echo**

Etiquetas para inserción de código

Dentro de dichas etiquetas aplicaremos la sintaxis de PHP. Las sentencias de código finalizan con “**;**”

- Los espacios, tabulaciones, intros, etc en el código embebido no tienen otro efecto que **mejorar legibilidad**
- Los scripts se pueden incrustar en cualquier sección del HTML y puede haber un número indefinido en el fichero.
- Además, debemos guardarlo con la extensión **.php** para que el servidor use el intérprete de PHP. Éste sustituye el script por el resultado de su ejecución, incluyendo las etiquetas de inicio y fin.

Uso de directivas – php.ini

Para configurar el entorno del Servidor para PHP, se modifica el fichero **php.ini** (intérprete de PHP)

- El fichero **php.ini** indica una serie de valores que determinan el comportamiento del intérprete PHP
- Se encuentra ubicado en el directorio raíz bajo los nombres **php.ini-development** y **php.ini-production**
- En Ubuntu lo encontramos en /etc/php/8.3/cli/php.ini (8.3 es la versión instalada, puede variar)
- Las instrucciones del fichero se denominan **Directivas**

Listado de directivas de php.ini

<https://www.php.net/manual/es/ini.list.php>

Uso de directivas – php.ini

- Las directivas se forman por una pareja de clave-valor.
- Las directivas que comienzan por **;** están comentadas y son ignoradas por el motor del intérprete.
- Para indicar las rutas dentro del fichero se utilizan los formatos:
`C:\directorio\directorio`
`\directorio\directorio`
`/directorio/directorio`
- El fichero php.ini se lee cada vez que se arranca el servidor web.
- El servidor busca el fichero php.ini por este orden:
 - ✓ En el propio directorio de php.
 - ✓ En la ruta definida como variable de entorno.
 - ✓ En el directorio del sistema (Ej: C:\Windows) que es la opción recomendada.

Ámbito de las variables

Es el contexto en el que se puede acceder a una variable.

- En PHP existen variables **locales** y **globales**.
- Las variables se definen como globales precediéndolas de la palabra **global**.
global \$var=5;
- También las podemos definir como globales asignándolas a la matriz superglobal **\$GLOBALS**.
\$GLOBALS[\$var]=5;

Ámbito de las variables

Si queremos mantener el valor de una variable local en las sucesivas llamadas a la función hay que definirla como ***static***

Salida del código:

01

12

23

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

Ámbito de las variables

```
<?php
function PruebaSinGlobal(){
    $var++;
    echo "Prueba sin global. \$var: ".$var."\n";
}
function PruebaConGlobal(){
    global $var;
    $var++;
    echo "Prueba con global. \$var: ".$var."\n";
}
function PruebaConGlobals(){
    $GLOBALS["var"]++;
    echo "Prueba con GLOBALS. \$var: ".$GLOBALS["var"]."\n";
}
//variable global
$var=20;
PruebaSinGlobal();
PruebaConGlobal();
PruebaConGlobals();
?>
```

Resultado:

```
Prueba sin global. $var: 1
Prueba con global. $var: 21
Prueba con GLOBALS. $var: 22
```

Constantes

- Una constante es un identificador de un dato que no cambia de valor durante toda la ejecución de un programa.
- Las constantes no se asignan con el operador **=**, sino con la función **define**:

```
define(nombre_constante_entre_comillas, dato_constante);  
define ("PI", 3.1416);  
print PI;
```

- No llevan \$ delante
- La función `defined("PI")` devuelve TRUE si existe la constante.
- Son siempre globales por defecto.
- Sólo se pueden definir constantes de los tipos escalares (boolean, integer, double, string)

Constantes Predefinidas

Dependen de las extensiones que se hayan cargado en el servidor, aunque hay constantes predefinidas que siempre están presentes:

- `PHP_VERSION`: Indica la versión de PHP que se está utilizando.
- `PHP_OS`: Nombre del sistema operativo que ejecuta PHP.
- `TRUE`
- `FALSE`
- `E_ERROR`: Indica los errores de interpretación que no se pueden recuperar.
- `E_PARSE`: Indica errores de sintaxis que no se pueden recuperar.
- `E_ALL`: Representa a todas las constantes que empiezan por `E_`.

Utilización de las variables

- Restricciones sobre las variables:
 - ✓ Deben comenzar por **\$**
 - ✓ Seguidamente debe haber una letra (may/min) o guión bajo (`_`)
 - ✓ El resto de caracteres pueden ser números, letras guiones bajos
- PHP es case sensitive
- Si una variable se compone de varias palabras, se aconseja escribirla en minúsculas excepto el inicio de la siguiente palabra (camelCase)
- Las **variables predefinidas** siguen el patrón **`$_VARIABLE`** por lo que se desaconseja usar este mismo patrón
- Las variables o cadenas se pueden concatenar usando el punto (`.`)

Variables

Además de las variables definidas por el programador, existen gran cantidad de *variables predefinidas* que se pueden usar libremente:

- ✓ ***Variables de entorno***: Variables que el servidor pone a disposición de PHP e indirectamente del programador
- ✓ ***Variables de PHP***: Variables predefinidas que pertenecen al intérprete PHP y que éste pone a disposición del programador.
- A partir de PHP 4 se incluyen matrices **superglobales** que centralizan todas las variables predefinidas.
\$GLOBALS, \$_SERVER, \$_GET, \$_POST, \$_COOKIE, \$_FILES, \$_ENV, \$_REQUEST, \$_SESSION

Variables Superglobales

- **\$_GET**: lleva los datos de forma "visible" al cliente (navegador web). El **medio de envío es la URL**. Para recoger los datos que llegan en la url se usa **\$_GET**.

Ejemplo: `www.midominio.com/action.php?nombre=silvia&apellidos1=vilar`

- **\$_POST**: consiste en datos "ocultos" (porque el cliente no los ve) **enviados por un formulario** cuyo método de envío es POST. Ideal para formularios. Para recoger los datos que llegan por este método se usa **\$_POST**.
- **\$_REQUEST**: Con la variable **\$_REQUEST** **recuperaremos los datos** de los formularios enviados tanto por **GET como por POST**.

Variables Superglobales \$_GET

Cadena de caracteres añadida a la URL:

`http://example.com/?nombre=Silvia&apellido=Vilar`

- Código PHP:

```
<?php
echo '¡Hola ' . $_GET["nombre"] . ' ' . $_GET["apellido"]!';
echo '¡Hola ' . $_REQUEST["nombre"] . ' ' .
$_REQUEST["apellido"]!';
?>
```

- Resultado:
¡Hola Silvia Vilar!

Variables Superglobales \$_POST y \$_REQUEST

A través de un formulario:

- Formulario HTML:

```
<form action="ejemplo.php" method="POST">  
  Nombre usuario: <input type="text" name="username" /><br />  
  Email: <input type="text" name="email" /><br />  
  <input type="submit" name="btnEnviar" value="Enviar" />  
</form>
```

- Código PHP:

```
<?php  
echo "¡Hola " . $_POST['username'] . "!";  
echo "¡Hola " . $_REQUEST['username'] . "!";  
?>
```

- Resultado:

```
¡Hola Silvia!  
¡Hola Silvia!
```


Variables Superglobales \$_COOKIE

\$_COOKIE: Recoge las variables pasadas al script actual mediante Cookies

- Ejemplo:

//creación de la cookie

```
<?php
    setcookie("nombre", 'Silvia', time()+3600);
?>
```

//obtención de la cookie

```
<?php
echo '¡Hola ' .
htmlspecialchars($_COOKIE["nombre"]) . '!';
?>
```

- Resultado

¡Hola Silvia!

Variables Superglobales \$_SERVER

\$_SERVER: contiene información de cabeceras, rutas, ubicaciones, etc. del script

- Ejemplo:

```
<?php  
echo $_SERVER['SERVER_NAME'];  
?>
```

- Resultado

www.example.com

Variables de Variables

Se pueden crear nombres de variables dinámicamente anteponiendo **\$\$** a una variable.

- La variable ***variable*** toma su nombre del valor de otra variable previamente declarada.
- Ejemplo:

```
<?php
$var = "uno";
$$var = "dos";
print ($var);    // produce el texto: "uno"
print ($uno);    // produce el texto: "dos"
print ($$var);   // produce el texto: "dos"
print (${ $var});
```

- A diferencia de las variables por referencia, se están creando dos variables distintas que ocupan direcciones de memoria distintas.

Variables de Variables

Se pueden crear nombres de variables dinámicamente anteponiendo \$\$ a una variable.

- Otro Ejemplo:

```
<?PHP
    $mensaje_es="Hola";
    $mensaje_en="Hello";
    $idioma = "en";
    $mensaje = "mensaje_" . $idioma;
    print $$mensaje;
?>
```

Devolvería Hello

Tipos de datos. Conversiones entre tipos

- PHP soporta los tipos de datos primitivos:
 - ✓ Números enteros
 - ✓ Números en coma flotante
 - ✓ Cadenas de caracteres
 - ✓ Booleanos
 - ✓ Objetos
 - ✓ Recursos
 - ✓ NULL
- El tipo de una variable no se suele especificar. Se decide en tiempo de ejecución en función del contexto y puede variar.

Tipos de datos. Conversiones entre tipos

- Números enteros: Enteros positivos y negativos
\$var = 20; \$var = -20; // asignación decimal
\$var = 024; \$var = -024; // asignación octal
\$var = 0x14; \$var = -0x14; // asignación hexadecimal
- Números en coma flotante: Permiten almacenar una parte fraccionaria.
\$var = 260.78;
\$var = 26078e-2;
- Booleanos: Pueden almacenar los valores True (1) y False (0).
- Recursos: Son valores especiales que hacen referencia a una información de estado o memoria de origen externo a PHP. Ejemplo: una conexión a BD

Tipos de datos. Conversiones entre tipos

Funciones de interés:

- La función **gettype()** devuelve el tipo de una variable
- Las funciones **is_type** comprueban si una variable es de un tipo dado:
is_array(), is_bool(), is_float(), is_integer(),
is_null(), is_numeric(), is_object(), is_resource(),
is_scalar(), is_string()
- La función **var_dump()** muestra el tipo y el valor de una variable. Es especialmente interesante con los arrays.

Tipos de datos. Conversiones entre tipos

Tipo string:

- Comillas simples o dobles (en este caso interpreta secuencias de escape como caracteres especiales)
- Otra forma de inicializar cadenas es utilizar la sintaxis heredoc (desde PHP 4) y nowdoc (desde PHP 5.3) que utilizan el símbolo de documento incrustado (“<<<”) y un identificador para marcar el final del documento.

Ver:

<https://www.php.net/manual/es/language.types.string.php#language.types.string.syntax.heredoc>

<https://www.php.net/manual/es/language.types.string.php#language.types.string.syntax.nowdoc>

NOTA: Es importante no escribir ningún carácter, salvo \n, antes y después del identificador de cierre de la cadena.

- Acceso a un carácter de la cadena: \$inicial=\$nombre{0};

Tipos de datos. Conversiones entre tipos

- Ejemplos de inicialización de cadenas:

```
$a = 9;
```

```
print 'a vale $a\n'; // muestra a vale $a\n
```

```
print "a vale $a\n"; // muestra a vale 9 y avanza una línea
```

```
print "<IMG SRC='logo.gif'>"; // muestra <IMG  
SRC='logo.gif'>
```

```
print "<IMG SRC=\"logo.gif\">"; //muestra <IMG  
SRC="logo.gif">
```

```
$nombre="Silvia";
```

```
$var = <<<xxx // Sintaxis heredoc con delimitador xxx
```

```
Esta es una cadena que termina al encontrarse xxx. $nombre  
xxx;
```

```
/* Resultado a mostrar: Esta es una cadena que termina al  
encontrarse xxx Silvia*/
```

Conversiones automáticas de tipos de datos.

- PHP es muy flexible en el manejo de los tipos de datos.
- PHP evalúa la operación a realizar y el tipo de los operandos, y adapta los operandos para poder realizar la operación lo más correctamente posible.
- En operaciones entre enteros y coma flotante el resultado es un número en coma flotante
- Una concatenación de cadenas con una variable numérica hace que ésta última sea convertida a cadena.

```
$varN=1;
```

```
$varC='4 flores';
```

```
$varC=$varN.$varC; // el resultado es 14 flores
```

Silvia Vilar Pérez

27

Sin embargo, `$varN=$varC+$varN` el resultado sería 5

Conversiones automáticas de tipos de datos.

Reglas automáticas de conversión de tipos:

- En operaciones lógicas, los datos NULL, 0, '0' y ' ' se consideran FALSE. Cualquier otro dato se considera TRUE (incluida la cadena 'FALSE').
- En operaciones aritméticas no unitarias las cadenas se intentan leer como números y, si no se puede, se convierten en 0, TRUE se convierte en 1, y FALSE se convierte en 0.
- En operaciones de comparación, si un operando es un número, el otro también se convertirá en un número. Sólo si ambos operandos son cadenas se compararán como cadena.
- En operaciones de cadenas de caracteres, NULL y FALSE se convierten en ' ', y TRUE se convierte en '1'.

Conversión forzada de tipos de datos.

- La conversión automática que realiza PHP no siempre es lo que queremos obtener.
- PHP permite otras conversiones implícitas de tipos :
 - (int) : Fuerza la conversión a entero
 - (real), (double), (float): Fuerza la conversión a coma flotante.
 - (string): Fuerza la conversión a cadena de caracteres.
 - (array): Fuerza la conversión a matriz
 - (object): Fuerza la conversión a un objeto.

Sintaxis básica PHP

- Comentarios (como en C):
 - ✓ `/* ... */` varias líneas
 - ✓ `//` una línea
 - ✓ `#` Comentario estilo shell para una línea
- Para imprimir: `echo` y `print`
 - ✓ **echo**: muestra una o más cadenas separadas por comas
 - `echo "Hola mundo";`
 - `echo "Hola ", "mundo";` //elementos separados
 - `echo "Hola " . "mundo";` //elementos concatenados
 - ✓ **print**: muestra una cadena o varias unidas por el operador punto (`.`)
 - `print "Hola " . "mundo";`
 - `print "Hola mundo";`

Sintaxis básica PHP

- Uso de `\n` para generar código HTML legible
 - ✓ Sin el carácter `\n`

Código PHP

```
print("<P>Párrafo 1</P>");  
print("<P>Párrafo 2</P>");
```

Código HTML

```
<P>Párrafo 1</P><P>Párrafo 2</P>
```

Salida

Párrafo 1

Párrafo 2

Sintaxis básica PHP

- Uso de \n para generar código HTML legible
 - ✓ Con el carácter \n

Código PHP

```
print ("<P>Párrafo 1</P>\n");  
print ("<P>Párrafo 2</P>\n");
```

Código HTML

```
<P>Párrafo 1</P>  
<P>Párrafo 2</P>
```

Salida

```
Párrafo 1  
  
Párrafo 2
```

Expresiones y Operadores

- Operadores aritméticos: +, -, *, /, %, ++, --
- Operador de asignación: =
- Operadores combinados: -=, +=, *=, /=, .=", %=
- Ejemplos:

```
$a = 3; $a += 5; // a vale 8  
$b = "hola ";  
$b .= "mundo"; // b vale "hola mundo"
```
- Operadores de comparación: ==, !=, <, >, <=, >= y otros

Expresiones y Operadores

- Operador de identidad **===** compara también el tipo de las variables.

- Operador de control de error: **@**

Antepuesto a una expresión, evita cualquier mensaje de error que pueda ser generado por la expresión y continua la ejecución

```
<?php
$var1=3; $var2=0;
$huboerror="no se produce resultado por error";
$nohuboerror="variable con valor";
@$resultado = $var1/$var2;
echo (empty($resultado))? huboerror : nohuboerror;
?>
```

- Operadores lógicos: && (and), || (or), ! , xor
- Operadores de cadena:
 - ✓ concatenación: **.** (punto)
 - ✓ asignación con concatenación: **.=**

Inclusión de Ficheros externos en PHP

- La inclusión de ficheros externos se consigue con:
 - ✓ **include()**
 - ✓ **require()**
- Ambos incluyen y evalúan el fichero especificado
- Diferencia: en caso de error **include()** produce un warning y **require()** un error fatal
- Se usará **require()** si al producirse un error debe interrumpirse la carga de la página
- Si usamos **include_once()** o **required_once()**, sólo lo realizará la primera vez

Inclusión de Ficheros externos en PHP

```
<HTML>
  <HEAD>
    <TITLE>Título</TITLE>
    <?PHP
      // Incluir bibliotecas de funciones
      require ("conecta.php");
      require ("fecha.php");
      require ("cadena.php");
      require ("globals.php");
    ?>
  </HEAD>
  <BODY>
    <?PHP
      include ("cabecera.html");
    ?>
    // Código HTML + PHP
    <?PHP
      include ("pie.html");
    ?>
  </BODY>
</HTML>
```



UD3. Estructuras en PHP

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Tomas de decisión. Estructuras de control
- Estructuras iterativas.
- Estructuras de control de flujo.
- Arrays
- Características de los Arrays
- Creación y eliminación de Arrays.
- Operaciones sobre Arrays
- Cadenas
- Funciones
- Manejo de Fecha y Hora
- Pruebas y Depuración

Tomas de decisión. Estructuras de control

Podemos usar las estructuras condicionales:

SENTENCIA SWITCH

```
<?php
switch ($i) {
    case 0:
        echo "i es igual a 0";
        break;
    case 1:
        echo "i es igual a 1";
        break;
    case 2:
        echo "i es igual a 2";
        break;
    default:
        echo "No se imprime si hay break"
}
?>
```

SENTENCIA IF

```
<?php
if ($a > $b) {
    echo "a es mayor que b";
} elseif ($a == $b) {
    echo "a es igual que b";
} else {
    echo "a es menor que b";
}
?>
```

Estructuras repetitivas

Podemos usar las estructuras repetitivas:

SENTENCIA WHILE

```
<?php
$i = 1;
while ($i <= 10) {
    echo $i;
    $i++;
}
```

```
?>
```

```
<?php
$i = 1;
while ($i <= 10):
    echo $i;
    $i++;
```

```
endwhile;
```

```
?>
```

SENTENCIA DO WHILE

```
<?php
do {
    if ($i < 5) {
        echo "i no es lo suficientemente grande";
        break;
    }
    $i = $i * $i;
    if ($i < $minimum_limit) {
        break;
    }
    echo "i está bien";
    /* procesar i */
} while (0);
?>
```

Estructuras iterativas

Podemos usar las estructuras iterativas:

SENTENCIA FOR

```
<?php
for ($i=0;$i<10; $i++){
    $suma=$suma+$i;

    echo "suma: $i \n";
}
?>
```

SENTENCIA FOREACH

```
<?php
foreach ($elementos as $e){
    echo "elemento: $e \n";
}

foreach ($elementos as
$key=>$value){
    echo "$key => $value \n";
}
?>
```


Estructuras de control de flujo

BREAK es la sentencia para salir directamente:

SENTENCIA BREAK

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

NOTA: El número entero opcional indica el número de niveles de la estructura anidada que queremos salir

Estructuras de control de flujo

CONTINUE permite abandonar la iteración y seguir:

SENTENCIA CONTINUE

```
<?php
$i = 0;
while ($i++ < 5) {
    echo "Exterior\n";
    while (1) {
        echo "Medio\n";
        while (1) {
            echo "Interior\n";
            continue 3;
        }
        echo "Esto nunca se imprimirá.\n";
    }
    echo "Ni esto tampoco.\n";
}
?>
```

NOTA: El número entero opcional indica el número de niveles de la estructura anidada que queremos saltar y volver a evaluar la condición tras ese salto

Arrays (Matrices)

- Un array almacena pares clave-valor
- Puede tener un número variable de elementos
- Cada elemento puede tener un valor de tipo simple (integer, string, etc.) o compuesto (otro array)
- El array que contiene otro/s arrays (matrices) se denomina multidimensional
- PHP admite:
 - ✓ **Arrays escalares** donde los *índices* son *números*
 - ✓ **Arrays asociativos** donde los *índices* son *cadenas* de caracteres

Arrays

Sintaxis:

`array ([clave =>] valor1, [clave =>] valor2, ...)`

- La clave es una cadena o un entero no negativo.
- El valor puede ser de cualquier tipo válido en PHP, incluyendo otro array
- Ejemplos:
`$color = array ('rojo'=>101, 'verde'=>51, 'azul'=>25);`
`$medidas = array (10, 25, 15);`
- Acceso a los arrays del ejemplo:
`$color['rojo']` // No olvidar las comillas
`$medidas[0]`
- El primer elemento es el 0

Características de los Arrays

Características

- Los arrays no se declaran, ni siquiera para indicar su tamaño (como el resto de las variables).
- Pueden ser dispersos (se implementan como tablas hash).
- Los índices de sus elementos no tienen porque ser consecutivos.
`$vec[1] = '1º elemento';`
`$vec[8] = '2º elemento';`
- En realidad contienen un mapeo entre claves y valores (arrays asociativos)
`array([index]=>[valor], [index2]=>[valor], ...);`

Características de los Arrays

Características

- Los índices no tienen porque ser números (arrays asociativos cuyo índice es una cadena de caracteres)

`$vec['tercero'] = '3º elemento';`

- Los arrays no son homogéneos. Sus elementos pueden ser de cualquier tipo (incluso tipo Array) y ser de tipos diferentes en el mismo vector.

`$vec[5] = '4º elemento';`

`$vec[1000] = 5.0;`

Creación de Arrays

Formas de crear arrays

- Asignación directa: Se añaden los elementos uno a uno indicando el índice entre []. Si no existía, se crea

```
$vec[5] = "1º elemento"; $vec[1] = "2º elemento";  
$vec[6] = "3º elemento"; $vec[ ] = "3º elemento"; //sin indicar  
clave toma el valor siguiente al máximo de los índices enteros
```
- Usando el constructor ***array()***.
 - ✓ Se añaden entre paréntesis los elementos. El índice comienza en 0 Ej:

```
$vec = array ( 2, 9.7, "Silvia");
```


//

```
$vec[0] = 2, $vec[1] = 9.7 y $vec[2] = "Silvia"
```
 - ✓ Se puede fijar el índice con el operador =>

```
$vec = array ( 4 => 2, 9.7, "nombre"=> "Silvia");
```


//

```
$vec[4] = 2, $vec[5] = 9.7 y $vec["nombre"] = "Silvia"
```

Eliminación de Arrays

- Para eliminar los elementos del array se usa ***unset()***
unset(\$vec[5]) unset(\$vec['nombre']) unset(\$vec)
// La última elimina el array completo
- Imprimimos el array y sus valores con ***var_dump(\$vec)*** o bien con ***print_r(\$vec)*** //no echo.
- Podemos reindexar el array para que su índice comience en 0 con ***array_values(\$vec)***

```
$vec[3] = 6;  
$vec[] = 7; //El índice valdría 4  
$array = array_values($array);  
print_r($vec);
```

Resultado:

```
Vec  
(  
    [0] => 6  
    [1] => 7  
)
```

Arrays Asociativos

La clave o índice en un String. Pueden definirse:

- Mediante la función array()

```
$precios = array("Azúcar" => 1, "Aceite" => 4, "Arroz" => 0.5);
```

```
$capitales = array("Francia"=>"París", "Italia"=>"Roma");
```

- Por referencia

```
$precios["Azúcar"] = 1;
```

```
$precios["Aceite"] = 4;
```

```
$precios["Arroz"] = 0.5
```

```
$capitales["Francia"]="París";
```

```
$capitales ["Italia"]="Roma";
```

Arrays Multidimensionales

- Son arrays en los que al menos uno de sus valores es otro array
- Pueden ser escalares o asociativos

```
$pais=array(  
    "espana"=>array(  
        "nombre"=>"España",  
        "lengua"=>"Castellano",  
        "moneda"=>"Euro"),  
    "uk" =>array(  
        "nombre"=>"UK",  
        "lengua"=>"Inglés",  
        "moneda"=>"Libra"));
```

Silvia Vilar Pérez

```
Resultado de var_dump($pais):  
array(2) {  
    ["espana"]=>  
    array(3) {  
        ["nombre"]=>  
        string(7) "España"  
        ["lengua"]=>  
        string(10) "Castellano"  
        ["moneda"]=>  
        string(4) "Euro"  
    }  
    ["uk"]=>  
    array(3) {  
        ["nombre"]=>  
        string(2) "UK"  
        ["lengua"]=>  
        string(7) "Inglés"  
        ["moneda"]=>  
        string(5) "Libra"  
    }  
}
```

Recorrido en Arrays

Podemos recorrer los elementos del array con bucles

```
$ciudades = array("París", "Madrid", "Londres");
```

- Mostrar el contenido del array (for)

```
for ($i=0;$i<count($ciudades); $i++){  
    echo $ciudades[$i]; echo "\n";  
}
```

- Mostrar el contenido del array (foreach)

```
foreach ($ciudades as $ciudad){  
    echo $ciudad; echo "\n";  
}  
foreach ($ciudades as $key=>$ciudad){ //si el vector es asociativo  
    echo "$key => $ciudad"; echo "\n";  
}
```

También disponemos de multitud de funciones de arrays

Silvia Vilar Pérez

<https://www.php.net/manual/es/ref.array.php>

Cadenas

- Echo y print no permiten formatear la salida
print \$variable == echo \$variable.
echo "hola1","hola2"; → admite parámetros
print ("hola1","hola2"); → error! → hacer print "hola1"."hola2"
- **sprintf** (igual que printf): Devuelve cadena formateada con el formato indicado.

% - un carácter de porcentaje literal. No se requiere argumento.

b - argumento tipo integer y número binario.

c - argumento tipo integer carácter con valor ASCII.

d - argumento tipo integer y número decimal (con signo).

f/F - argumento tipo float y número de punto flotante (local/no local)

s - argumento tipo string.

Ejemplo: \$dia= 5; \$mes=3; \$anno=12;

printf("%**02**d/%**02**d/%**04**d", \$dia, \$mes, \$anno); (cantidad caracter/cifra)

Escribe: 05/03/0012

Cadenas

- Ejemplo

```
<?php
$s = 'mono';
$t = 'muchos monos';
printf("[%s]\n", $s); // salida estándar de string
printf("[%10s]\n", $s); // justificación a la derecha con espacios
printf("[% -10s]\n", $s); // justificación a la izquierda con espacios
printf("[%010s]\n", $s); // relleno con ceros también funciona con strings
printf("[% '#10s]\n", $s); // utiliza el carácter de relleno personalizado '#'
printf("[%10.10s]\n", $t); // justificación a la izquierda pero con un corte a los 10 caracteres
?>
```

- Resultado

```
[mono]
[  mono]
[mono  ]
[000000mono]
[#####mono]
[muchos mon]
```

Funciones

- Sintaxis:

```
function nombreFunción (param1,param2){  
    Instrucción1;  
    Instrucción2;  
    return valor_de_retorno;  
}
```

- Ejemplo:

```
function suma ($x, $y){  
    $s = $x + $y;  
    return $s;  
}
```

//La invocamos

```
$a=1;  
$b=2;  
$c=suma ($a, $b);  
print $c;
```

Funciones

- Todas las funciones y clases de PHP tienen ***ámbito global***. Se pueden llamar desde fuera de una función incluso si fueron definidas dentro, y viceversa.

```
<?php
function externa()
{
    function interna()
    {
        echo "No existo hasta que se llame a externa().\n";
    }
}
/* No podemos llamar aún a interna() ya que no existe. */
externa();
/* Ahora podemos llamar a interna(), la ejecución de externa() la ha
hecho accesible. */
interna();
?>
```

Funciones

- Los argumentos se pueden pasar por valor (\$i) o referencia (&\$i). En caso de tener argumentos con valor predeterminado, deben ir a la derecha de los no predeterminados

```
<?php
function hacer_yogur($sabor, $tipo = "acidófilo")
{
    return "Hacer un tazón de yogur $tipo de $sabor.\n";
}
echo hacer_yogur("frambuesa");
echo hacer_yogur("frambuesa", "dulce");
?>
```

Resultado:

Hacer un tazón de yogur acidófilo de frambuesa.
Hacer un tazón de yogur dulce de frambuesa.

Funciones Variables

- Funciones variables: si llamamos a una variable con paréntesis buscará una función con ese nombre y la ejecutará

```
<?php
function prueba($arg = ' '){
echo "Estamos en la función prueba() y el argumento es '$arg'.<br>\n";
}
$func = "prueba"; //inicializamos la variable
$func('hola'); // Esto llama a prueba('hola')
?>
```

Resultado:

Estamos en la función prueba() y el argumento es 'hola'

Ver <https://www.php.net/manual/es/functions.variable-functions.php>

Funciones Anónimas

- Las funciones anónimas, también conocidas como cierres (closures), permiten la creación de funciones que no tienen un nombre especificado. Se implementan usando la clase Closure. PHP 7.4 introduce funciones de **flecha** con sintaxis más concisa: **fn(list_args) => expr;**

```
<?php
$saludo = function($nombre)
{
    printf("Hola %s\n", $nombre);
}; //la variable $saludo contiene la declaración de la función
```

```
$saludo1 = fn($nombre) => printf("Hola %s\n", $nombre); //Función flecha (se obtiene el mismo resultado)
```

```
$saludo('Mundo');
$saludo1('PHP');
?>
```

Resultado:
Hola Mundo
Hola PHP

Ver <https://www.php.net/manual/es/functions.anonymous.php>

Funciones flecha: <https://www.php.net/manual/es/functions.arrow.php>

Manejo de Fecha y Hora

Clase DateTime

- Clase para trabajar con fechas y horas en PHP
- Debemos definir, en primer lugar, nuestra Zona Horaria:
 - ✓ En la sección Date del archivo php.ini
 - ;[Date]
 - ;Defines the default timezone used by the date functions
 - date.timezone = Europa/Madrid
 - ✓ Durante la ejecución con la función ***date_default_timezone_set()***. Esta función genera un error si contradice la configuración del php.ini.
- Listado de zonas horarias soportadas:
<https://www.php.net/manual/es/timezones.php>

Pruebas

Podemos aplicar distintas pruebas al SW:

- **Pruebas Unitarias:** En los módulos, se aplican a las funciones, estructuras de decisión, control de flujo, etc. usadas dentro de él.
- **Pruebas de Integración:** Se valida la interacción entre módulos a través de las interfaces, comunicación entre los mismos, etc.
- **Pruebas de Validación:** se aplican cuando se comprueba el cumplimiento de requisitos por la aplicación (pruebas alfa[programador-usuario] y beta[usuario])
- **Pruebas de Sistema:** Se aplican con el sistema en funcionamiento (producción). Ejemplo: pruebas de seguridad, rendimiento, recuperación, etc

Herramientas de Pruebas y depuración

Durante el desarrollo de la aplicación web, es necesario realizar las tareas de pruebas y depuración de código del programa. Para ello, básicamente nos centraremos en las siguientes herramientas disponibles para PHP:

- **PHP Unit:** Herramienta para poder diseñar y ejecutar las pruebas unitarias
- **XDebug:** Herramienta para poder depurar nuestro código, integrándola en el IDE que hayamos elegido para desarrollar código (Visual Studio Code en nuestro caso)

Herramienta PHPUnit

- Instalación: <https://phpunit.de/getting-started/phpunit-10.html>
 - 1) Abrimos terminal en nuestro proyecto y ejecutamos;
wget https://phar.phpunit.de/phpunit-10.phar
chmod +x phpunit-10.phar
 - 2) Comprobamos con **./phpunit-10.phar --version**
- En Visual Studio Code podemos instalar la extensión PHP Unit
- Normalmente crearemos en nuestro proyecto una carpeta test o tests donde guardaremos los test unitarios.
- Los test deben guardarse con nombre terminado en ***Test.php**

PHP Unit – Ejemplo Calculadora

```
<?php    /* Calculadora.php */
class Calculadora
{
    public function sumar($a=0, $b=0)
    {
        return $a + $b;
    }
    public function restar($a=0,$b=0)
    {
        //
    }
    public function multiplicar($a=1,$b=1)
    {
        //
    }
    public function dividir($a=1,$b=1)
    {
        //
    }
}
?>
```

PHP Unit – Ejemplo CalculadoraTest

```
<?php    /* CalculadoraTest.php */

use Calculadora;
use PHPUnit\Framework\TestCase;

class CalculadoraTest extends TestCase
{    //El nombre de las funciones de pruebas debe comenzar por test*
    public function testSumar()
    {
        $cal = new Calculadora();
        $this->assertEquals( 6, $cal->sumar(2,4), "2+4 debe dar 6" );
        // más assertEquals tests...
    }
}
?>
```


Herramienta PHP Unit - Ejemplo

- Podemos ejecutar las pruebas en el terminal de Visual Studio Code con la instrucción:

`./phpunit-10.phar --bootstrap ./src/Calculadora.php test`

- Añadiendo el directorio test, ejecuta todos los test de la carpeta, si queremos que ejecute sólo un test en concreto lo indicamos:

`./phpunit-10.phar --bootstrap ./src/Calculadora.php ./test/CalculadoraTest.php`

- La ejecución de la instrucción nos dará el resultado de las pruebas:

PHPUnit 10.3.5 by Sebastian Bergmann and contributors.

Runtime: PHP 8.1.2-1ubuntu2.14

.

1 / 1 (100%)

Time: 00:00.001, Memory: 24.46 MB

OK (1 test, 1 assertion)

Silvia Vilar Pérez

Herramienta Xdebug - Install

- Si no lo tenemos ya instalado, visitaremos <https://xdebug.org/wizard.php>, pegaremos el resultado de `phpinfo()` y nos indicará la versión adecuada para la versión de PHP instalada así como las instrucciones para instalarlo.
- Debemos actualizar `php.ini` con las siguientes directivas al final del fichero (a partir de xDebug 3.0):

```
[XDebug]
```

```
xdebug.remote_enable = 1
```

```
xdebug.start_with_request = yes
```

```
xdebug.idekey="vscode"
```

```
zend_extension = /*la ubicación que se indique en la  
instalación*/ (.so en Ubuntu o .dll en Windows)
```

Herramienta XDebug

- Para activar la depuración ejecutaremos **Run → StartDebugging** y seleccionaremos **PHP**.
- Lo que ejecutemos a partir de entonces se detendrá en los puntos de ruptura que hayamos establecido.
- Nos saldrá una pequeña barra con opciones para depurar paso a paso, pausar, reiniciar, parar, etc.
- Si queremos realizar una traza de las pruebas, nos debemos asegurar de activar el debugging, indicar los puntos de ruptura y en el terminal de Visual Studio Code ejecutamos las pruebas con la instrucción indicada en PHPUnit (`./phpunit-10.phar`)



Institut d'Educació Secundària
Paiporta

UD4. Formularios

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Formularios: atributos
- Elemento LABEL
- Elemento FIELDSET (agrupa controles)
- Elemento TEXTAREA (caja texto múltiples líneas)
- Elementos de tipo INPUT
- Elemento SELECT (desplegable)
- Servidor interno PHP

Atributos de Form

Los formularios nos permiten recoger datos del usuario para enviarlos al servidor y que éste los procese.

La etiqueta **<form>** tiene los siguientes atributos:

- **action**: Indica el fichero en el servidor al que se enviarán los datos del formulario. Si se omite, es la página actual
- **method**: indica el método HTTP a usar para enviar los datos. Por defecto es GET
- **autocomplete**: Permite (o no) completar con datos introducidos anteriormente por el usuario. Por defecto su valor es “on”
- **name**: especifica el nombre del formulario
- **novalidate**: es un atributo booleano que, si se incluye, indica que los datos del formulario no deben ser validados al enviarse

Elementos de Form

Podemos encontrar varios elementos en un formulario:

- Elemento LABEL
- Elemento FIELDSET (agrupa controles)
- Elemento TEXTAREA (caja texto múltiples líneas)
- Elementos de tipo INPUT
 - ✓ TEXT (caja de texto de una única línea)
 - ✓ PASSWORD
 - ✓ HIDDEN (control oculto para obtener datos)
 - ✓ RADIO (botón selección)
 - ✓ CHECKBOX (check de selección)
 - ✓ BUTTON
 - ✓ SUBMIT
 - ✓ RESET
 - ✓ FILE (para selección de archivos a enviar)
- Elemento SELECT (desplegable)
 - ✓ Simple (sólo un elemento)
 - ✓ Múltiple (varios elementos a la vez)

LABEL

El elemento **<label>** define una etiqueta para otros elementos. Ayuda a describir el el input que debe rellenar el usuario

Con radio button o checkbox, se marca la opción al pichar encima de la etiqueta

Su atributo **for** debe tener el valor del atributo **id** del input al que acompaña para que se puedan correlacionar

Label:

```
<label for="nombre">Nombre:</label>
```

```
<input type="text" id="nombre" name="nombre">
```

FIELDSET

El elemento **<fieldset>** es un contenedor que agrupa diversos controles que tienen datos relacionados

El elemento **<legend>** define la leyenda o texto para el grupo de controles

```
<form action="ejemplo.php">
```

```
<fieldset>
```

```
<legend>Datos Personales:</legend>
```

```
<label for="nombre">Nombre:</label><br>
```

```
<input type="text" id="nombre" name="nombre"><br>
```

```
<label for="apellido">Apellido:</label><br>
```

```
<input type="text" id="apellido" name="apellido"><br>
```

```
<input type="submit" value="Enviar">
```

```
</fieldset>
```

```
</form>
```

A diagram illustrating the visual representation of a fieldset. It consists of a rectangular box with a thin border. The word "Fieldset" is written in a blue, serif font at the top left of the box. A horizontal line extends from the top right corner of the box, and a vertical line extends from the bottom right corner, suggesting the box is part of a larger form layout.

TEXTAREA

El elemento **TEXTAREA** muestra un campo de texto multilínea (área de texto)

- Ejemplo

Comentario:

```
<TEXTAREA COLS="50" ROWS="4" NAME="comentario">
```

Este libro me parece ...

```
</TEXTAREA>
```

- Se obtienen los datos del campo en PHP:

```
<?PHP
```

```
$comentario = $_REQUEST['comentario'];
```

```
echo $comentario;
```

```
?>
```

INPUT TEXT

El elemento INPUT de tipo **TEXT** muestra un campo de texto de una sola línea

- Ejemplo:

Introduzca la cadena a buscar:

```
<INPUT TYPE="text" NAME="cadena" VALUE="valor por defecto" SIZE="20">
```

- Se obtienen los datos del campo en PHP:

```
<?php  
$cadena = $_REQUEST['cadena'];  
echo $cadena;  
?>
```



INPUT PASSWORD

El elemento INPUT de tipo **PASSWORD** muestra un campo de texto de una sola línea en el cual los caracteres se ocultan al escribirlos (normalmente puntos)



- Ejemplo

Contraseña:

```
<INPUT TYPE="password" NAME="clave">
```

- Se obtienen los datos del control en PHP:

```
<?PHP
```

```
$clave = $_REQUEST['clave'];
```

```
echo $clave;
```

```
?>
```


INPUT HIDDEN

El elemento INPUT de tipo **HIDDEN** es un control que no se muestra al usuario (oculto) y se usa para incluir datos que no pueden ser vistos ni modificados por el usuario.

- Ejemplo

```
<INPUT TYPE='hidden' NAME='username'  
VALUE="$usuario">
```

- Se obtienen los datos del control en PHP:

```
<?PHP  
$username = $_REQUEST['username'];  
echo $username;  
?>
```

INPUT RADIO

El elemento INPUT de tipo **RADIO** muestra radio button para selección exclusiva de SÓLO UNO de los elementos a elegir, por lo que los controles tienen el mismo nombre y distintos valores. La opción por defecto se indica con **checked**

- Ejemplo:

Radio: ☒ opción 1 ☐ opción 2

Sexo:

```
<INPUT TYPE="radio" NAME="sexo" VALUE="M"
CHECKED>Mujer
```

```
<INPUT TYPE="radio" NAME="sexo" VALUE="H">Hombre
```

- Se obtienen los datos del control en PHP:

```
<?PHP
```

```
$sexo = $_REQUEST['sexo'];
echo ($sexo);
?>
```

INPUT CHECKBOX

El elemento INPUT de tipo **CHECKBOX** muestra casillas de verificación para selección CERO o MÁS elementos a elegir, por lo que los controles tienen el mismo nombre y los valores se almacenan en un array. La/s opción/es marcadas por defecto se indican con **checked**

- Ejemplo

```
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="garaje"
CHECKED>Garaje
```

```
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="piscina">Piscina
```

```
<INPUT TYPE="checkbox" NAME="extras[]" VALUE="jardin">Jardín
```

- Se obtienen los datos del control en PHP:

```
<?PHP
```

```
$extras = $_REQUEST['extras'];
```

```
foreach ($extras as $extra)
```

```
echo "$extra<br>\n";
```

```
?>
```

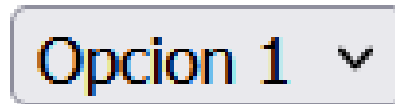
☒ opción 1 ☒ opción 2 ☐ opción 3

SELECT SIMPLE

El elemento **SELECT SIMPLE** muestra un desplegable con valores para poder seleccionar UN ÚNICO valor. Los valores se indican dentro del control y la opción seleccionada por defecto se indica con **selected**

- Ejemplo

Color:



```
<SELECT NAME="color">  
<OPTION VALUE="rojo" SELECTED>Rojo</OPTION>  
<OPTION VALUE="verde">Verde</OPTION>  
<OPTION VALUE="azul">Azul</OPTION>  
</SELECT>
```

- Se obtienen los datos del control en PHP:

```
<?PHP  
$color = $_REQUEST['color'];  
echo $color;  
?>
```

SELECT MÚLTIPLE

El elemento **SELECT MÚLTIPLE** muestra un desplegable con valores para poder seleccionar VARIOS de modo que se almacenan en un array. La/s opción/es seleccionada/s por defecto se indican con **selected**. El atributo **Size** indica el número de valores a mostrar cada vez

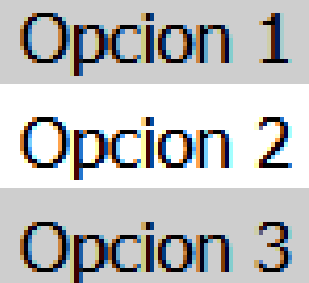
- Ejemplo

Idiomas:

```
<SELECT MULTIPLE SIZE="2" NAME="idiomas[]">
<OPTION VALUE="ingles" SELECTED>Inglés</OPTION>
<OPTION VALUE="frances" SELECTED>Francés</OPTION>
<OPTION VALUE="aleman">Alemán</OPTION>
</SELECT>
```

- Se obtienen los datos del control en PHP:

```
<?PHP
$idiomas = $_REQUEST['idiomas'];
foreach ($idiomas as $idioma)
echo "$idioma<br>\n";?>
```



INPUT BUTTON

El elemento INPUT de tipo **BUTTON** muestra un botón.

- Ejemplo

```
<INPUT TYPE="button" NAME="saludo"  
ONCLICK="alert('Hola PHP') VALUE="saludo">  
<INPUT TYPE="button" NAME="actualizar"  
VALUE="Actualiza datos">
```



- Se obtienen los datos del control en PHP:

```
<?PHP  
$saludo = $_REQUEST['saludo'];  
if ($saludo) { //Si no es cadena vacía es true  
    print ("Se ha visualizado el saludo");}  
$actualizar = $_REQUEST['actualizar'];  
if ($actualizar=="Actualiza datos") {  
    print ("Se han actualizado los datos");}  
?>
```


INPUT SUBMIT

El elemento INPUT de tipo **SUBMIT** muestra un botón para enviar los datos del formulario al destino indicado en el action del formulario.



- Ejemplo

```
<INPUT TYPE=submit NAME="enviar" VALUE="Enviar datos">
```

- Se obtienen los datos del control en PHP:

```
<?PHP
```

```
$enviar = $_REQUEST['enviar'];
```

```
if ($enviar=="Enviar datos") { //También funciona if($enviar)
```

```
    echo "Se ha pulsado el botón de enviar";
```

```
}
```

```
?>
```

INPUT SUBMIT - formaction

El elemento INPUT de tipo **SUBMIT** tiene atributos para poder modificar el comportamiento del formulario. Con **formaction** especifica una URL para su acción distinta a la que indica el formulario cuando se procesa, esto es, sobrescribe el atributo **action** del **form**

- Ejemplo

```
<form action="pagina1.php">
```

```
Nombre:<input type="text" name="nombre"><br>
```

```
Apellidos:<input type="text" name="apellidos"><br>
```

```
<input type="submit" value="Enviar">
```

```
<input type="submit" formaction="pagina2.php"
```

```
value="Enviar como Admin">
```

```
</form>
```

INPUT RESET

El elemento INPUT de tipo **RESET** muestra el formulario en blanco “limpiando” los controles.



- Ejemplo

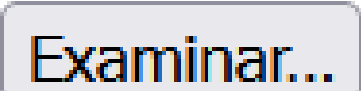
```
<INPUT TYPE=reset NAME="borrar" VALUE="Limpiar datos">
```

- Se obtienen los datos del control en PHP:

```
<?PHP
$borrar = $_REQUEST['borrar'];
if($borrar=="Limpiar datos") { //También if($borrar)
    echo "Se ha pulsado el botón de limpiar datos";
}
?>
```

INPUT FILE

El elemento INPUT de tipo **FILE** muestra un control para anexar un archivo de modo que podemos seleccionarlo y subirlo con el botón Examinar. Para poder subir archivos, el **formulario** debe tener **method="POST"** y **enctype="multipart/form-data"**

- Ejemplo:  No se ha seleccionado ningún archivo.

```
<FORM ACTION="procesa.php" METHOD="post"  
ENCTYPE="multipart/form-data">
```

```
<INPUT TYPE="file" NAME="fichero">  
</FORM>
```

- Este tipo se detalla en la siguiente unidad con la subida de archivos al servidor

Servidor interno PHP

PHP-CLI facilita un servidor interno de PHP con fines de desarrollo y depuración.

Algunos parámetros interesantes son:

- t**; especifica la raíz de los documentos de forma explícita
- s**: Genera código HTML que muestra la sintaxis de ficheros PHP coloreada
- S**: Crea un servidor web que sirve/ejecuta código PHP

Si no se especifica un fichero, PHP busca el fichero `index.php` o `index.html` en su defecto

Servidor interno PHP

Para crear el servidor interno, nos situaremos en la carpeta donde tenemos almacenados los ficheros de PHP

En el terminal ejecutaremos:

php -S localhost:8000 //Puede ser otro puerto

En el navegador, introduciremos la url

127.0.0.1:8000 o bien **localhost:8000**

En el terminal podemos ver la comunicación entre cliente y servidor y posibles datos de depuración de la ejecución del código PHP



UD5. Procesamiento de Formularios

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

- Procesamiento de Formularios
- Métodos GET y POST
- Subida de archivos al servidor
- Validación de formularios
- Comprobación de datos
- Expresiones Regulares
- Cabeceras HTTP

Protocolo HTTP

- El protocolo HTTP es un protocolo sin estado (no guarda información o estado del cliente) → [usar cookies o variables ocultas en formularios]. La comunicación dura lo necesario para resolver la petición del cliente.
- A partir de HTTP/1.1 tiene un mecanismo keep-alive para que la conexión pueda ser reutilizada para más de una petición.

Formularios

- Los Formularios no forman parte de PHP sino del lenguaje estándar de Internet: HTML.
 - Puesto que se utiliza el protocolo HTTP, estamos limitados por su interfaz, sólo se puede utilizar algunos de los comandos del protocolo para establecer la comunicación: GET o POST
 - Dos tipos diferentes de peticiones, según atributo method del <FORM>:
 - ✓ Peticiones GET (método GET de HTTP)
 - ✓ Peticiones POST (método POST de HTTP)
- <FORM ACTION="nombreFichero.php" METHOD="POST/GET">**
- Al pulsar el botón de envío el navegador construye la petición adecuada

Formularios - Peticiones GET

Los parámetros se indican en la URL tras el signo “?” y se concatenan con & indicando variable=valor.

- En el servidor, los valores se guardan en el array asociativo **\$_GET**. La URL que se genera es similar a:

http://site/procesa.php?name1=value1&name2=value2

- Reglas de codificación URL:
 - ✓ RFC 3986 ([Sección 2: Caracteres](#))
 - ✓ Los caracteres especiales se codifican con el formato %NN (NN: valor hexadecimal de carácter). El servidor se encarga de decodificarlo
- Son caracteres especiales:
 - ✓ Ñ, ñ, á, etc. (no tienen un carácter US ASCII asociado)
 - ✓ No seguros: “<” “>” “”” (delimitadores url), “#” (secciones), “%” (permite codificar caracteres)
 - ✓ Reservados: “/” “@” “&” “?” “:” “[” “]” “!” “\$” “'” “(” “)” “*” “+” “,” “;” “=”

Peticiones GET - URL

A continuación se muestra una tabla con la codificación de los caracteres ASCII más comunes a UTF-8:

Caracter original	Caracter Codificado	Caracter original	Caracter Codificado
/	%2F	?	%3F
:	%3A	@	%40
=	%3D	&	%26
“	%22	\	%5C
‘	%60	~	%7E
(espacio en blanco)	%20	#	%23

Peticiones GET - URL

A continuación se muestra una tabla con la codificación de los caracteres ASCII no ingleses más comunes a UTF-8:

Caracter original	Caracter Codificado	Caracter original	Caracter Codificado
ñ	%C3%B1	Ñ	%C3%91
á	%C3%A1	Á	%C3%81
é	%C3%A9	É	%C3%89
í	%C3%AD	Í	%C3%8D
ó	%C3%B3	Ó	%C3%93
ú	%C3%BA	Ú	%C3%9A
ç	%C3%A7	Ç	%C3%87

Peticiones GET - URL

Ejemplo:

```
echo "<a href=\"proces.php?user=$user&uid=$uid\">";
```

- Si \$user="Álvaro Gil" y \$uid="12&57" genera el enlace:
 // INCORRECTO
- Se pueden usar las funciones php urlencode (el espacio es +) y rawurlencode (el espacio lo codifica) (**RFC 3986**)
echo "<a href=\"proces.php?user=" . urlencode(\$user) . "&uid=" .
urlencode(\$uid) . "\">";

Esto genera:

```
<a href="proces.php?user=%C3%81lvaro+Gil&uid=12%2657">  
//CORRECTO
```

Ejercicio: Codificar la url de user Cándido García Sánchez y uid 25@12#3. Probad con ambas funciones

```
C%C3%A1ndido+Garc%C3%A1a+S%C3%A1nchez&uid=25%4012%233
```

Formularios - Peticiones POST

Los parámetros se envían en el cuerpo del mensaje (no url).

- El servidor, almacena los valores en el array asociativo **\$_POST**. Los caracteres especiales se traducen a ASCII.
- Es necesario indicar en el **<form>** el tipo de codificación para enviar datos al servidor con el atributo **enctype**:
 - ✓ **application/x-www-form-urlencoded** (Por defecto). Los caracteres se codifican antes de ser enviados (espacios se convierten en “+” y caracteres especiales a %NN). NO PERMITE ENVIAR ARCHIVOS
 - ✓ **multipart/form-data** No se codifican caracteres, Se requiere en forms que envían ficheros. PERMITE ENVIAR ARCHIVOS
 - ✓ **text/plain** Los espacios se convierten a “+” pero los caracteres especiales no se codifican

Formularios – POST VS GET

- Problemas GET
 - ✓ No se puede enviar información binaria (archivos, imágenes, etc.) => necesario el método POST.
 - ✓ Los datos se ven en la URL del navegador.
- Problemas POST
 - ✓ Rompe la funcionalidad del botón “Atrás” del navegador
 - ✓ El botón actualizar repite la operación
- Recomendaciones generales
 - ✓ GET implica “obtener” información.
 - ✓ POST implica “realizar” una acción con un “efecto secundario” (procesar información, almacenarla, etc.).
 - ✓ Mejor POST para procesar formularios

Formularios – Resumen

El acceso a los valores introducidos en los formularios se realiza a través de arrays globales:

- **\$_GET**: parámetros enviados mediante método GET, se envían en la URL
- **\$_POST**: parámetros enviados mediante el método POST se envían en el cuerpo del mensaje HTTP
- **\$_REQUEST**: array asociativo que contiene los datos de \$_GET y \$_POST

Formularios – HTTP GET vs POST

El método **GET** de HTTP:

- Añade los datos en la URL en pares nombre=valor
- La longitud de la URL es limitada a 2048 caracteres
- Útil para datos no seguros como strings de consultas en google
- Útil en envíos de formulario donde el usuario quiere guardar em marcadores o favoritos el resultado

El método **POST** de HTTP:

- Añade los datos del formulario en el cuerpo de la petición http (no se muestran en la URL)
- No tiene limitación de tamaño, puede usarse para enviar grandes cantidades de datos
- Los envíos de formularios con POST no pueden guardarse en marcadores o favoritos

Acceso a controles de formulario desde PHP – Ejemplo

Acceso a los datos de un formulario HTML:

- Fichero uno.html

```
<html><body>
<form action="dos.php" method="POST">
    Edad: <input type="text" name="edad">
    <input type="submit" value="aceptar">
</form>
</body></html>
```

- Fichero dos.php

```
<?php
echo "La edad es:". $_POST['edad'];
// o bien echo "La edad es:". $_REQUEST['edad'];
?>
```

Acceso a controles de formulario desde PHP – Ejercicio

- Formulario HTML:

```
<form action="respuesta.php?valor=10" method="POST">  
<input type="text" name="nombre">  
</form>
```

- PHP (¿qué obtenemos en cada caso?)

```
<?
```

```
echo 'valor: ' . $_GET['valor'] . '<br>'; valor=10
```

```
echo 'valor: ' . $_POST['valor'] . '<br>'; vacío, no ha obtenido datos
```

```
echo 'valor: ' . $_REQUEST['valor'] . '<br>'; valor=10
```

```
echo 'nombre: ' . $_GET['nombre'] . '<br>'; vacío, no ha obtenido datos
```

```
echo 'nombre: ' . $_POST['nombre'] . '<br>'; nombre=introducido por user
```

```
echo 'nombre: ' . $_REQUEST['nombre'] . '<br>'; nombre=introducido por user  
>
```

Subida de archivos al servidor

- Para subir un fichero al servidor se utiliza el input **FILE**
- Hay que tener en cuenta:
 - ✓ El elemento FORM debe tener el atributo ENCTYPE="multipart/form-data"

```
<form action="upload.php" method="post" enctype="multipart/form-data">
```

- ✓ El fichero tiene un límite en cuanto a su tamaño. Este límite se fija de dos formas diferentes y complementarias:
 - En el fichero de configuración php.ini
 - En el propio formulario

Subida de archivos al servidor

Límite de tamaño en php.ini

```
.....  
.....  
; File Uploads ;  
.....  
.....  
; Si se permite o no subir archivos mediante HTTP  
file_uploads = On  
;  
; Tamaño máximo de cada archivo subido.  
upload_max_filesize = 2M  
;  
; Tamaño máximo de los datos mandados por POST  
;(incluidos los que no sean archivos)  
post_max_size = 8M
```

Subida de archivos al servidor

Límite de tamaño en el propio formulario

```
<!-- MAX_FILE_SIZE must precede the file input field -->  
<input type="HIDDEN" name="MAX_FILE_SIZE"  
value='102400'>  
<input type="FILE" name="fichero">
```

Ejemplo:

```
<input type="FILE" size="44" name="imagen">
```

- MAX_FILE_SIZE se usa en un input oculto para manejar la validación de tamaño en el cliente
- El valor se indica en bytes mediante un valor entero

Subida de archivos al servidor

La variable **\$_FILES** contiene toda la información del fichero subido:

- `$_FILES['imagen']['name']`: Nombre original del fichero en el cliente
- `$_FILES['imagen']['type']`: Tipo MIME del fichero. Por ejemplo, "image/gif"
- `$_FILES['imagen']['size']`: Tamaño en bytes del fichero subido
- `$_FILES['imagen']['tmp_name']`: Nombre temporal del fichero que se genera para guardar el fichero subido
- `$_FILES['imagen']['error']`: Código de error asociado a la subida del fichero

Subida de archivos al servidor

Aspectos a tener en cuenta:

- Debe comprobarse que el fichero se ha subido correctamente: **is_uploaded_file("nombre_temp_de_\$_FILES")**
- Una vez comprobado, debe darse al fichero un nombre único. Por ello, y como norma general, se descarta el nombre original del fichero y se crea uno nuevo añadiéndole, por ejemplo, la fecha y hora.
- El fichero subido se almacena en un directorio temporal y tenemos que moverlo al directorio de destino usando la función **move_uploaded_file()**

move_uploaded_file (\$_FILES['imagen'] ['tmp_name'], \$destino)

Los posibles errores en la subida del fichero los obtenemos en **UPLOAD_ERR**.

Ver <https://www.php.net/manual/es/features.file-upload.php>

Subida de archivos al servidor - Ejemplo

Código del formulario

```
<html>
```

```
<body>
```

Inserción de la fotografía del usuario:

```
<form action="inserta.php" method="post" enctype="multipart/form-data">
```

```
<?php
```

```
echo "Nombre usuario:<input type='text' name='usuario'/><br/>";
```

```
echo "Fotografía:<input type='file' name='imagen'/><br/>";
```

```
?>
```

```
<input type="submit" value="Enviar">
```

```
</form>
```

```
</body>
```

```
</html>
```

Código de inserta.php

```
<html><body><?php
    echo "name:".$_FILES['imagen']['name']."\n";
    echo "tmp_name:".$_FILES['imagen']['tmp_name']."\n";
    echo "size:".$_FILES['imagen']['size']."\n";
    echo "type:".$_FILES['imagen']['type']."\n";
    if (is_uploaded_file ($_FILES['imagen']['tmp_name'] )){
        $nombreDirectorio = "img/";
        $nombreFichero = $_FILES['imagen']['name'];
        $nombreCompleto = $nombreDirectorio.$nombreFichero;
        if (is_dir($nombreDirectorio)){ // es un directorio existente
            $idUnico = time();
            $nombreFichero = $idUnico."-".$nombreFichero;
            $nombreCompleto = $nombreDirectorio.$nombreFichero;
            move_uploaded_file ($_FILES['imagen']['tmp_name'],$nombreCompleto);
            echo "Fichero subido con el nombre: $nombreFichero<br>";
        }
        else echo 'Directorio definitivo inválido';
    }
    else
        print ("No se ha podido subir el fichero\n");
?></body></html>
```

Procesamiento de un formulario en único fichero

- Una forma de trabajar con formularios en PHP es utilizar un único fichero que procese el formulario o lo muestre según haya sido o no enviado respectivamente
- Ventajas:
 - ✓ Disminuye el número de ficheros
 - ✓ Permite validar los datos del formulario en el propio formulario
- Procedimiento:
 - si se ha enviado el formulario
 - Procesar formulario
 - si no
 - Mostrar formulario
 - fsi

La primera vez que carga la página se muestra el formulario. La segunda vez que carga, se procesa el formulario

Procesamiento de un formulario en único fichero

Para saber si se ha enviado el formulario se acude a la variable correspondiente al botón de envío.

Si este botón aparece de la siguiente forma en el formulario HTML:

```
<INPUT TYPE=SUBMIT NAME="enviar" VALUE="procesar">
```

entonces la condición anterior se transforma en:

```
if (isset($_POST['enviar']))
```

o bien

```
if ($_POST['enviar'] == "procesar")
```

Validación de formularios

- Toda información recibida de un formulario debe considerarse por norma contaminada y hay que validarla antes de darla por correcta y procesarla
- Lo más eficiente es mostrar los errores sobre el propio formulario para facilitar su corrección. Procedimiento:

si se ha enviado el formulario

Validar datos

si hay errores

Mostrar formulario con errores

si no

Procesar formulario

fsi

si no

Mostrar formulario con valores por defecto o ya enviados

fsi

Validación formulario no vacío

Una vez sabemos que el formulario se ha enviado para su validación, debemos comprobarlos siguiente:

1. Los campos del formulario no estén vacíos
2. Los tipos de los datos insertados en los campos sean los esperados (no etiquetas html, por ejemplo)
3. Que los datos sean introducidos correctamente (sin blancos, sin caracteres especiales, etc.)

NOTA: en validaciones al comparar usad `===` que devolverá true sólo si ambos valores son iguales y además del mismo tipo de dato (que ambos sean números, cadenas de texto, etc.)

Validación - Ejemplo

Valida que hay datos: Función **isset(\$variable)**. *Devuelve true si la variable está definida y no es NULL y false en otro caso*

Tenemos un formulario con un check de aceptar. Cuando está seleccionado, \$_REQUEST tiene la referencia al dato pero en otro caso no hay un índice definido:

```
<?php
if (isset($_REQUEST["acepto"])) { //variable definida
    print "<p>Desea recibir información</p>\n";
} else { //variable no definida o NULL
    print "<p>No desea recibir información</p>\n";
}
?>
```

Validación - Ejemplo

Valida que el usuario no ha introducido etiquetas html:
Función **strip_tags(\$variable)**. *Elimina las etiquetas HTML y PHP de la variable*

Ante la entrada de texto Silvia<Vilar>

El código:

```
<?php  
print "<p>Su nombre es " . strip_tags($_REQUEST["nombre"]) .  
"</p>\n";  
?>
```

Obtendría el resultado

<p>Su nombre es Silvia</p> //Vilar no lo mostraría por identificarlo como etiqueta

Validación - Ejemplo

Valida que no hay espacios en blanco en el texto: Función **trim(\$variable)**. *Elimina los espacios en blanco antes y después del texto.*

Ante una entrada de usuario con espacios en blanco

El código

```
<?php
if (trim($_REQUEST["nombre"]) == "") {
    print "<p>No ha escrito ningún nombre</p>\n";
} else {
    print "<p>Su nombre es ". trim($_REQUEST["nombre"]) . "</p>\n";
}
?>
```

Devolvería: <p>No ha escrito ningún nombre</p>

Validación – Ejemplo

Sustituir caracteres especiales por entidades HTML: Función **htmlspecialchars(\$variable)**: *Devuelve \$variable sustituyendo los caracteres especiales por entidades HTML*

Con la entrada de nombre Silvia & <Vilar>

El código

?php

```
$salida = htmlspecialchars($nombre, ENT_QUOTES, "UTF-8");
```

```
echo $salida;
```

```
?>
```

Devolvería
Silvia & <Vilar>

Fichero de validación de datos

- También puede ser útil generar un fichero `valida.php` donde se reúnan distintas validaciones y sea incluido en los ficheros php que procesan los datos del formulario con **`include`**, **`require`**, **`include_once`** o **`require_once`** (**`_once`** comprueba si se ha incluido ya previamente)
- De este modo, al incluir el fichero ***valida.php*** en el del procesamiento de los datos, disponemos de todas las funciones de validación para invocarlas y así trabajar con datos validados
- Tras validar los datos sin obtener errores, podríamos redirigir al usuario a la página ***validado.php*** con ***header('Location: valido.php');***

Ejemplo valida.php

```
<?php
function validaRequerido($valor){ //Obliga a introducir datos en campos requeridos
    if(trim($valor) == ""){
        return false;
    }else{
        return true;
    }
}
function validarEntero($valor, $opciones=null){ //valida que se haya introducido un
número entero
    if(filter_var($valor, FILTER_VALIDATE_INT, $opciones) === FALSE){
        return false;
    }else{
        return true;
    }
}
function validaEmail($valor){ //valida que se haya introducido un email
user@ejemplo.com
    if(filter_var($valor, FILTER_VALIDATE_EMAIL) === FALSE){
        return false;
    }else{
        return true;
    }
}
?>
```

```

<?php
    /* FICHERO INDEX.PHP*/
    require_once 'funciones/valida.php'; //Importamos el archivo con las validaciones (requerido, y lo carga una
    vez).
    //Guarda los valores de los campos en variables, siempre y cuando se haya enviado el formulario, si no guardará
    NULL
    $nombre = isset($_POST['nombre']) ? $_POST['nombre'] : null;
    $edad = isset($_POST['edad']) ? $_POST['edad'] : null;
    $email = isset($_POST['email']) ? $_POST['email'] : null;
    $errores = array(); //Este array guardará los errores de validación que surjan.
    //Pregunta si está llegando una petición por POST, lo que significa que el usuario envió el formulario.
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        if (!validaRequerido($nombre)) { //Valida que el campo nombre no esté vacío.
            $errores[] = 'El campo nombre es incorrecto.';
        }
        $opciones_edad = array(
            'options' => array( //Definimos el rango de edad entre 3 a 130.
                'min_range' => 3,
                'max_range' => 130
            )
        );
        if (!validarEntero($edad, $opciones_edad)) { //Valida la edad con un rango de 3 a 130 años.
            $errores[] = 'El campo edad es incorrecto.';
        }
        if (!validaEmail($email)) { //Valida que el campo email sea correcto.
            $errores[] = 'El campo email es incorrecto.';
        }
        //Verifica si ha encontrado errores y de no haber redirige a la página con el mensaje de que pasó la validación.
        if(!$errores){
            header('Location: validado.php');
            exit;
        }
    }
}
?>

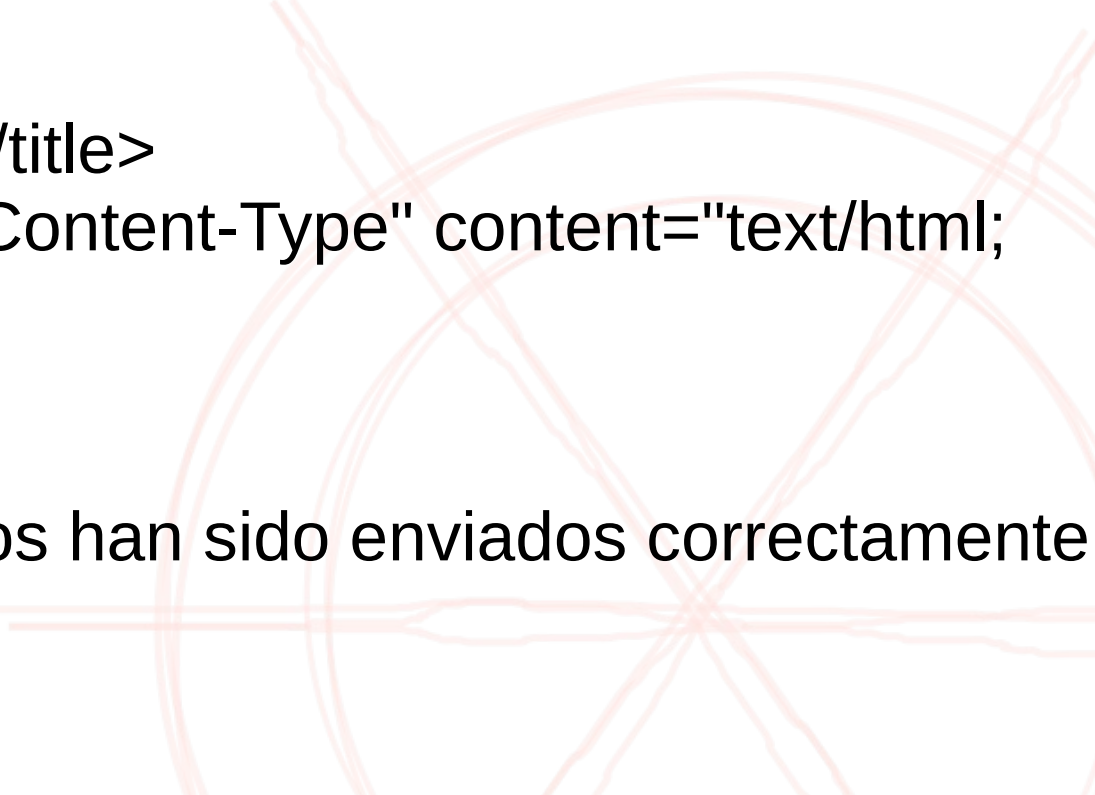
```

index.php (continuación)

```
<!DOCTYPE html>
<html>
  <head>
    <title> Formulario </title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <?php if ($errores): ?>
      <ul style="color: #f00;">
        <?php foreach ($errores as $error): ?>
          <li> <?php echo $error ?> </li>
        <?php endforeach; ?>
      </ul>
    <?php endif; ?>
    <form method="post" action="index.php">
      <label> Nombre </label><br />
      <input type="text" name="nombre" value="<?php echo $nombre ?>" /><br />
      <label> Edad </label><br />
      <input type="text" name="edad" size="3" value="<?php echo $edad ?>" /><br />
      <label> E-mail </label><br />
      <input type="text" name="email" value="<?php echo $email ?>" /><br />
      <input type="submit" value="Enviar" />
    </form>
  </body>
</html>
```

Ejemplo validado.php

```
<!DOCTYPE html>
<html>
  <head>
    <title> Formulario </title>
    <meta http-equiv="Content-Type" content="text/html;
charset=utf-8" />
  </head>
  <body>
    <strong> Sus datos han sido enviados correctamente
  </strong>
</body>
</html>
```



Funciones para comprobación de datos

PHP dispone de las siguientes funciones para la comprobación de datos:

- Funciones **is_** (Comprueba si la variable es del tipo indicado)
- Funciones **ctype_** (Comprueban si se corresponden los caracteres con el juego de caracteres local)
- Funciones **filter_** (Aplican filtros de comprobación)
- Funciones **_exists** (Indica si existe el objeto para el que se invoca)
- **Expresiones regulares** (Permiten aplicar patrones de coincidencia a las cadenas de texto)

Funciones is_

isset() pertenecería a este grupo de funciones

	Funcion	Tipo de datos que comprueba
Existencia	is_null(\$valor)	Devuelve true si es NULL
Números	is_bool(\$valor)	Devuelve si es booleano
	is_numeric(\$valor)	Devuelve si es número (puede ser negativo, con parte decimal y en decimal, hexadecimal, etc.)
	is_int(\$valor)	Devuelve si es entero
	is_float(\$valor)	Devuelve si es float (decimal)
Cadenas	is_string(\$valor)	Devuelve si es cadena
Otros	is_scalar(\$valor)	Devuelve si es escalar (entero, float, cadena o booleano)
	is_array(\$valor)	Devuelve si es un vector o matriz
	is_object(\$valor)	Devuelve si es un objeto

Funciones ctype_

Funcion	Tipo de datos que comprueba
ctype_alnum(\$valor)	Devuelve true si es alfanumérico (letras o números)
ctype_alpha(\$valor)	Devuelve si es carácter del alfabeto local [ES_es.UTF-8](incluyendo mayúsculas, minúsculas, acentos, ñ, ç, etc)
ctype_cntrl(\$valor)	Devuelve si es carácter de control (salto de línea, tabulador, escape, etc.)
ctype_digit(\$valor)	Devuelve si es dígito numérico
ctype_graph(\$valor)	Devuelve si es carácter imprimible (excepto espacios)
ctype_lower(\$valor)	Devuelve si es minúscula
ctype_print(\$valor)	Devuelve si carácter imprimible
ctype_punct(\$valor)	Devuelve si es signo de puntuación (carácter imprimible no alfanumérico ni espacio en blanco)
ctype_space(\$valor)	Devuelve si es espacio en blanco (espacios, tabuladores, saltos de línea, etc)
ctype_upper(\$valor)	Devuelve si es mayúscula
ctype_xdigit(\$valor)	Devuelve si es dígito hexadecimal

Funciones Filter_

Se usa la función **filter_var(\$valor [, \$filtro [, \$opciones]])** que devuelve los datos filtrados o false si el filtro falla

Filtros de validación	Datos que valida
FILTER_VALIDATE_BOOLEAN	Devuelve true para “1”, “true”, “on y “yes”. Falso en otro caso
FILTER_VALIDATE_INT	Valida un valor como integer, opcionalmente desde el rango especificado, y lo convierte a int en caso de éxito.
FILTER_VALIDATE_EMAIL	Valida una dirección de correo electrónico
FILTER_VALIDATE_FLOAT	Valida si es un float
FILTER_VALIDATE_IP	Valida una dirección IP con opciones IPv4, IPv6, etc,
FILTER_VALIDATE_MAC	Valida una dirección MAC
FILTER_VALIDATE_URL	Valida una URL no internacionalizada (RFC2396)
FILTER_VALIDATE_REGEXP	Valida expresiones regulares compatibles con Perl

Funciones _exists

Funciones de existencia	Datos que valida
file_exists (\$ruta)	Devuelve true si existe el fichero o directorio
class_exists(\$class_name)	Verifica si la clase ha sido definida o no
method_exists(\$objeto,\$metodo)	Comprueba si existe el método de la clase en el objeto (instancia o clase) indicado
property_exists(\$clase,\$propiedad)	Comprueba si la propiedad existe en la clase indicada
function_exists(\$funcion)	Comprueba si la función existe en las funciones definidas, las incluidas(internas) y las definidas por el usuario
array_key_Exists(\$indice,\$array)	Comprueba si el índice existe en el array. No funciona en claves anidadas de arrays multidimensionales

Expresiones Regulares

- Las expresiones regulares permiten definir **patrones de coincidencia** y aplicarlas a cadenas de texto para saber si la cadena (o parte de ella) cumple el patrón e incluso realizar transformaciones de la cadena.
- Para comprobar si una cadena cumple un patrón se usa la función `preg_match()` que devuelve 1 si coincide la cadena con el patrón, 0 si no coincide y FALSE si ha habido un error

`preg_match($patron, $cadena [, $matriz_coincidencias [, $modificadores [, $desplazamiento]]])`

Busca en `$cadena` una coincidencia con el `$patron`. En `$matriz_coincidencias` se almacena, comenzando en 0, el texto que coincide con el patrón completo y con índices superiores los subpatrones de coincidencia. Con el modificador, **PREG_OFFSET_CAPTURE** se guarda el índice de la posición de coincidencia encontrada en la cadena

Ejemplo preg_match()

preg_match(\$patron, \$cadena [, \$matriz_coincidencias [, \$modificadores [, \$desplazamiento]])

```
<?php
preg_match('/(foo)(bar)(baz)/', 'foobarbaz', $matches,
PREG_OFFSET_CAPTURE);

print_r($matches);
?>
```

Nomenclatura en ExpReg:
'/' - Limitador de expresión regular
() - agrupación

```
Array
(
    [0] => Array
        (
            [0] => foobarbaz
            [1] => 0
        )
    [1] => Array
        (
            [0] => foo
            [1] => 0
        )
    [2] => Array
        (
            [0] => bar
            [1] => 3
        )
    [3] => Array
        (
            [0] => baz
            [1] => 6
        )
)
```

Nota: PHP usa Perl Compatible Regular Expressions (PCRE)

Ejemplo preg_match()

```
<?php
$cadena = "Esto es una cadena de prueba";
$patron = "/de/";
$encontrado = preg_match_all($patron, $cadena,
$coincidencias, PREG_OFFSET_CAPTURE);

if ($encontrado) {
    print "<pre>"; print_r($coincidencias); print "</pre>\n";
    print "<p>Se han encontrado $encontrado
coincidencias.</p>\n";
    foreach ($coincidencias[0] as $coincide) {
        print "<p>Cadena: '$coincide[0]' - Posición:
$coincide[1]</p>\n";
    }
} else {
    print "<p>No se han encontrado coincidencias.</p>\n";
}
?>
```

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => de
                    [1] => 14
                )
            [1] => Array
                (
                    [0] => de
                    [1] => 19
                )
        )
)
```

Se han encontrado 2 coincidencias.
Cadena: 'de' - Posición: 14
Cadena: 'de' - Posición: 19

Nomenclatura en ExpReg
'/' - Limitador de expresión regular

Otras Funciones con Expresiones Regulares

Además de **preg_match()** que busca la coincidencia de la cadena con el patrón tenemos las siguientes funciones:

- **preg_filter()**: Realiza una búsqueda y sustitución de una expresión regular
- **preg_grep()**: Devuelve entradas de matriz/vector que coinciden con el patrón
- **preg_matchl()**: Busca todas las coincidencias de la cadena con el patrón.
- **preg_match_all()**: Busca todas las coincidencias de la cadena con el patrón y tras la primera coincidencia, las búsquedas continúan desde el final de dicha coincidencia
- **preg_quote()**: Escapa los caracteres de sintaxis de una expresión regular que son **. \ + * ? [^] \$ () { } = ! < > | : -**. Si se especifica delimiter, este carácter también se escapa
- **preg_replace()**: Busca en la cadena coincidencias de patrón y las reemplaza con replacement
- **preg_split()**: Divide un string usando una expresión regular

Expresiones Regulares

Podemos crear y comprobar nuestras expresiones regulares con ayuda de las siguientes webs:

<https://regex101.com/>

<https://www.regextester.com/>

<https://www.debuggex.com/>

<https://regexr.com/>

Expresiones Regulares - Ejemplos

```
<?php
```

```
// Devuelve true si "abc" se encuentra en cualquier lugar de $cadena.
```

```
preg_match("/abc/", $cadena); // abc aabc ahhjabckl abcerwabc
```

```
// Devuelve true si "abc" se encuentra al comienzo de $cadena.
```

```
preg_match("/^abc/", $cadena); // abc aabc ahhjabckl abcerwabc
```

```
// Devuelve true si "abc" se encuentra al final de $cadena.
```

```
preg_match("/abc$/", $cadena); // abc aabc ahhjabckl abcerwabc
```

```
// Pone una etiqueta <br /> al principio de $cadena.
```

```
$cadena = preg_replace("^", "<br />", $cadena); // <br />abcerwabc\n
```

```
// Pone una etiqueta <br /> al final de $cadena.
```

```
$cadena = preg_replace("$", "<br />", $cadena); // abcerwabc\n<br />
```

```
// Se deshace de cualquier carácter de nueva línea en $cadena.
```

```
$cadena = preg_replace("\n", "", $cadena); // abcerwabc
```

```
?>
```

Cabeceras HTTP

- En los mensajes HTTP, la cabecera y el cuerpo del mensaje **se separan por una línea en blanco**.
- **Cliente:** En la cabecera se incluyen datos como versión HTTP, URI, método usado (GET/POST), navegador, idiomas, codificación de caracteres, cookies, etc. En el cuerpo incluye información a intercambiar con el servidor. Con GET el cuerpo va vacío y con POST incluye los datos del formulario
- **Servidor:** En la cabecera se incluyen datos como versión HTTP, Fecha, Servidor Web, código de status y texto asociado al mismo (403=Forbidden, 404=Not Found, 500=Internal Server Error, etc.), Tipo de contenido, Set-cookie, etc. En el cuerpo envía la página HTML al cliente

Cabeceras HTTP

- PHP puede generar cabeceras HTTP con **header("cabecera:valor");**

- Ejemplos:

```
header("location: http://www.upv.es");
```

```
header("HTTP/1.0 404 Not Found");
```

```
header("Pragma: no-cache");
```

Otras: Cache-Control, Expires, Last-Modified, etc.

- Es importante que sea **invocada antes de mostrar nada** por pantalla: etiquetas HTML, include() o require() con líneas en blanco desde un fichero o desde PHP.
- Las cabeceras HTTP pueden también modificar el comportamiento del navegador que recibe la respuesta

Ver <https://www.php.net/manual/es/function.header.php> y <http://www.faqs.org/rfcs/rfc2616.html>

Cabeceras HTTP

- PHP puede obtener las cabeceras de petición HTTP con **`apache_request_headers()`**;

- Ejemplo:

```
<?php
$headers = apache_request_headers();
foreach ($headers as $header => $value) {
    echo "$header: $value <br />\n";
}
?>
```

```
Host: 127.0.0.1:8000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
```


Cabeceras HTTP – Usos frecuentes

- Redirigir al cliente a otra dirección
`header ('Location: acceso_no_autorizado.php');`
- Mostrar un mensaje y redirigir al cliente a otra dirección
`header ('Refresh: 5; url=http://www.google.es');`
`echo 'Lo que busca no existe, le redirigiremos a Google en 5 segundos'`
- Ocultar la versión de nuestro intérprete PHP
(También se puede asignar valor 0 a la directiva `expose_PHP` de `php.ini`)
`header('X-Powered-By: adivina-adivinanza');`

Cabeceras HTTP – Usos frecuentes

- Ofrecer la descarga de un archivo desde PHP:
header('Content-type: **application/pdf**'); //Vamos a mostrar un pdf
header('Content-Disposition: **attachment**; filename="downloaded.pdf") //Lo llamaremos downloaded.pdf
readfile('original.pdf'); //El original o fuente del pdf para guardarlo
- Generar contenidos diferentes a páginas HTML con PHP:
Imágenes, documentos PDF, etc. Posible gracias a librerías de PHP como GD, PDFlib, Ming, etc.
header("Content-type:image/jpeg");
header("Content-Disposition:**inline** ; filename=captcha.jpg");
readfile(micaptcha.jpg);

Attachment fuerza la descarga del archivo con nombre downloaded.pdf aunque lea el original
Inline lo muestra en el navegador