Desarrollo Web en Entorno Servidor

2º DAW - Curso 2018-2019

Creación de páginas web dinámicas

Programación en entorno servidor

En la introducción ya hemos visto en que arquitecturas y tecnologías se apoyan tanto las páginas web como las aplicaciones web.

En este punto nos centraremos exclusivamente en la programación de páginas web en entorno servidor, en este caso con lenguaje PHP. También se les conoce como páginas dinámicas puesto que el contenido que presentan al usuario puede cambiar en función de las acciones de éste, no directamente porque el programador modifique su programación (como ocurre con las páginas estáticas). Así, en las páginas web dinámicas el usuario puede interactuar con el sitio web y modificar así la forma en que la información se presenta y también la propia información, puesto que puede gestionarla.

En este curso, hemos diferenciado además entre lo que se conoce como páginas web y aplicaciones web. Actualmente cada vez están más solapados los conceptos puesto que, el concepto de lo que se conocía antes como web ha ido cambiado. Así, sitios web como *Facebook*, *Emodo* o *Twitter* son auténticas aplicaciones (nos referimos siempre a sus sitios web) que funcionan directamente en el navegador web, por lo que pueden ser tratadas, claramente, como aplicaciones web. Por otro lado, tenemos sitios como blogs, periódicos online, . . ., que, aunque son dinámicos, se limitan practicamente a mostrar información al usuario (que puede interactur en mayor o menor medida). En este caso podemos hablar de páginas web (dinámicas). Eso si, entre unas y otras hay un sinfin de situaciones intermedias y hacen díficil decidir que hace que un sitio web pase a ser una aplicación web o viceversa.

Aunque este tema solamente trate sobre desarrollo en el lado servidor, obviamente necesitaremos trabajar también con HTML y CSS si queremos darle una apariencia minimamente correcta a nuestras páginas web a la hora de ser presentadas por el navegador. Para ello usaremos Bootstrap [http://getbootstrap.com] que es una librería de estilos para la creación rápida de sitios web. Además, en su web, podemos encontrar numerosos ejemplos sobre como estructurar una web. De esa manera, no tendremos que perder demasiado tiempo escribiendo y estructurando la parte visible de las webs y podremos centrarnos en la

programación del lado servidor.

En cualquier caso, por si fuera necesario, en la sección Referencias se pueden encontrar guías de referencia de HTML y CSS, por si fuera necesario consultarlas.

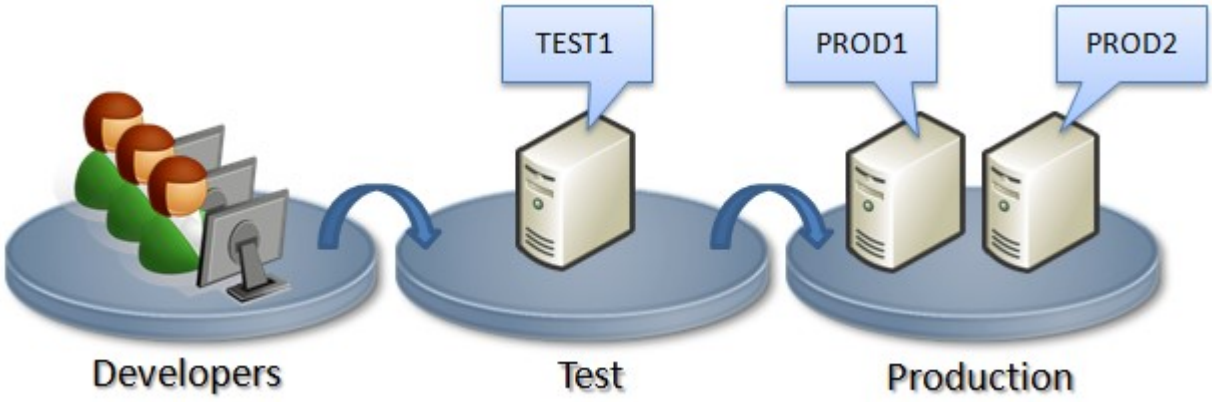
Instalación y configuración de PHP

Para la instalación y configuración de PHP, dispondremos de dos entornos:

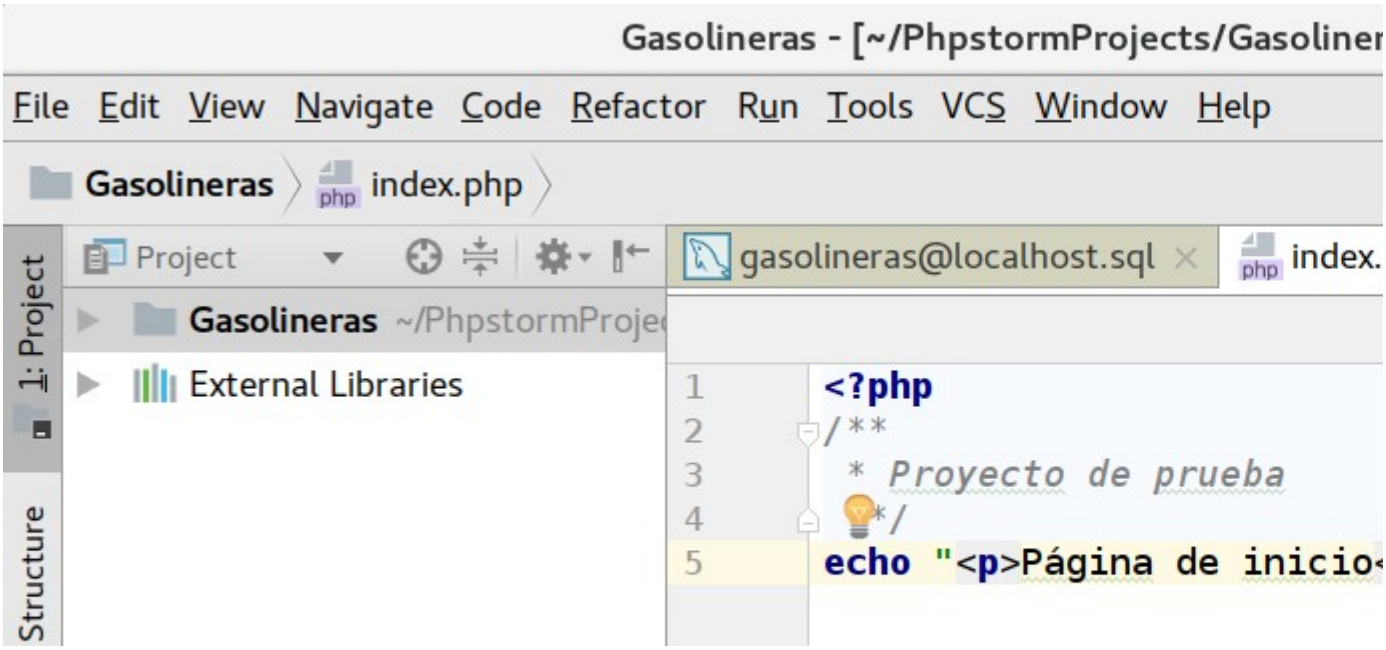
- **Entorno de desarrollo**. Será el entorno en el que trabajaremos en clase, directamente en nuestro equipo utilizando algún tipo de solución LAMP (**L**inux **A**pache **M**ySQL **P**HP). En la sección Software se pueden encontrar en enlace para la descarga de *XAMPP* si se va a trabajar en entornos Windows/OSX. En este caso simplemente se inicia el instalador y se siguen las instrucciones para la instalació de todo el software necesario. Si se va a trabajar en Linux, se puede utilizar la herramienta `apt` para la instalación del software necesario:

```
santi@zenbook:$ apt-get install mysql-server mysql-client apache2 php7.0
                php-mysql phpmyadmin
```

- **Entorno de producción**. Para este caso dispondremos de una máquina virtual Linux con el sistema LAMP instalado (ver punto anterior) cuya finalidad será aprender a montarlo como si fuera nuestro servidor de producción. Trabajaremos en el equipo local en nuestro entorno de desarrollo y cuando queramos subir los cambios a producción, lo haremos a esta máquina, que habremos instalado y configurado nosotros mismos. Además, dispondremos de acceso a una cuenta en un servidor remoto para poder colgar allí todos los ejercicios y proyectos que hagamos durante el curso. Esta cuenta estará ya configurada a través de CWP (CentOS Web Panel).



Entorno Integrado de Desarrollo: PhpStorm



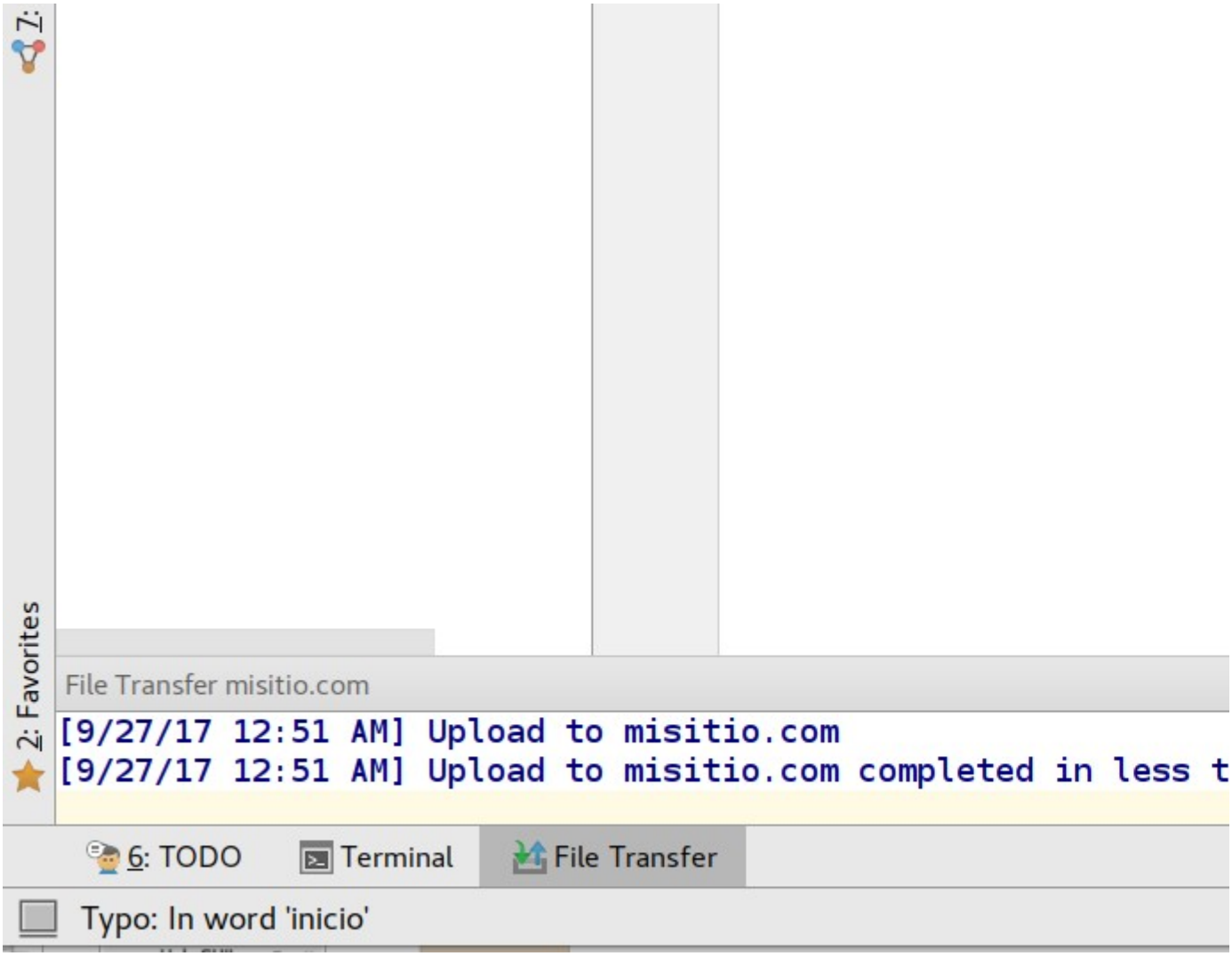


Figure 1: IDE PhpStorm

Despliegue de aplicaciones web con PhpStorm

Estructura de un script en PHP

A continuación se puede ver un ejemplo de script PHP en el que sólo encontramos código en ese lenguaje. Es por eso que el fichero lo almacenaremos con extensión *.php*

funcion.php

```
<?php
/* Esto es un comentario
   de varias lineas */
// Esto es un script PHP
function sumar($numero1, $numero2) {
    $resultado = $numero1 + $numero2;
    return $resultado;
}

$suma_total = sumar(10, 15)
echo "La suma es $suma_total";
```

También podemos tener ficheros que contengan tanto código PHP como HTML.

index.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primera página web</title>
  </head>
  <body>
    <?php echo "<p>Esta es mi primera página web</p>"; ?>
    <?php echo "<p>Hoy es " . date("d.m.Y") . "</p>"; ?>
    <p><?= "Esto es PH" ?></p>
    <a href="#">Esto será un enlace</a>
    <img src="#" alt="Esto será una imagen"/>
  </body>
</html>
```

Estructura

- Si el fichero contiene código PHP deberemos ponerle la extensión `.php` obligatoriamente para que Apache lo sepa y se lo haga ejecutar al intérprete
- Como se puede observar, todo script de PHP se inicia con la etiqueta `<?php` y no es necesario finalizar con etiqueta de cierre siempre y cuando el script sólo contenga código PHP. Si el fichero contiene PHP junto con HTML tendremos también que cerrar los bloques del primero con la etiqueta `?>`
- También se puede usar la etiqueta `<?= ?>` que imprime por pantalla el valor constante o variable que se escriba entre las dos etiquetas (equivale a la instrucción `echo`)
- La sintaxis del lenguaje es muy similar a Java. Más adelante iremos viendo alguna pequeñas diferencias
- Se puede interrumpir la ejecución de un script PHP con las funciones `exit(mensaje)` o `die()`

Organización del código en un proyecto PHP

- Podemos organizar el código en diferentes scripts PHP según nos convenga y reutilizar ese código en otros scripts con las directivas `include(script.php)`, `include_once(script.php)`, `require(script.php)` o `require_once(script.php)`. Las cuatro funciones permiten reutilizar el código que está escrito en el nombre del script que se pasa como parámetro pero presentan alguna diferencia

```
<?php
// Si falla produce un mensaje de WARNING
include("funciones.php");
/* Funciona como la función include() pero
   comprueba que el fichero no se haya incluido ya
 */
include_once("funciones.php");
// Si falla produce un mensaje de ERROR y detiene el script
require("funciones.php");
/* Funciona como la función require() pero
   comprueba que el fichero no se haya incluido ya
 */
require_once("funciones.php");
```

Estas funciones nos permitirán organizar nuestro sitio web de forma que aquellas partes que siempre se repiten o

necesitemos utilizar en diferentes páginas puedan ser importadas y no tengamos que repetirlas.

Supongamos un sitio web compuesto de varios documentos (en principio HTML) como este donde cabecera y pie de la página se mantienen y sólo cambiaremos el contenido principal de la web. Probablemente incluso haya algún menú superior y/o lateral que también se mantenga o simplemente cambie para mostrar la sección en la que nos encontramos.

index.html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Título de la web</title>
    <link rel="stylesheet" type="text/css" href="css/bootstrap.css"/>
</head>
<body>
<br/>
<div class="container">
    <h1>Información de actualidad</h1>
    <div class="list-group">
        <a href="noticias.html" class="list-group-item">Noticias</a>
        <a href="deportes.html" class="list-group-item">Deportes</a>
        <a href="television.html" class="list-group-item">Televisión</a>
        <a href="series.html" class="list-group-item">Series</a>
    </div>
    <div class="container">
        <p>Aqui irá el contenido de la web</p>
        <p>Y podrán ir muchas cosas dependiendo de la sección en la que esté</p>
    </div>
    <footer>
        &copy; 2017 Ejercicio
    </footer>
</div>
</body>
</html>
```

Está claro que siguiendo este esquema acabaremos teniendo varios documentos HTML (para noticias, deportes, television y series, al menos) que serán bastante parecidos al menos en cuanto a la maquetación de la web se refiere. Si que tendrán diferente contenido pero gran parte del código HTML se repetirá.

Así, si utilizamos PHP con su función `include` o `require` podremos organizar el código de una forma más óptima desde el punto de vista del desarrollador (el visitante nunca notará la diferencia) lo cual es muy importante sobre todo cuando el proyecto toma un tamaño considerable.

index.php

```
<?php
if (isset($_REQUEST["id"])
  $id = $_REQUEST["id"];
else
  $id = "noticias";
?>
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Título de la web</title>
  <link rel="stylesheet" type="text/css" href="css/bootstrap.css"/>
</head>
<body>
<br/>
<div class="container">
  <h1>Información de actualidad</h1>
  <div class="list-group">
    <a href="?id=noticias" class="list-group-item">Noticias</a>
    <a href="?id=deportes" class="list-group-item">Deportes</a>
    <a href="?id=television" class="list-group-item">Televisión</a>
```

```
      <a href="?id=series" class="list-group-item">Series</a>
    </div>
    <div class="container">
  <?php
    include($id . ".php");
  ?>
    </div>
    <footer>
      &copy; 2017 Ejercicio
    </footer>
  </div>
  </body>
  </html>
```

Hemos aprovechado la posibilidad de enviar parámetros en la URL para construir los hipervínculos de forma que apunten siempre a la misma web (en este caso `index.php`) pero con un valor diferente para el parámetro `id`. De esta forma, accediendo a dicho parámetro a través de la variable `$_REQUEST`, podemos leerlo y cargar la sección que corresponda en cada caso.

Con esta estructura, los diferentes documentos que contengan el código de cada sección simplemente tendrán el contenido de la propia sección puesto que se incluyen dentro de una estructura que mantiene un maquetado, cabecera y pie común para todo el sitio web, por lo que el código del sitio web queda mucho más organizado y el mantenimiento de la web mucho más sencillo (imaginemos que simplemente queremos cambiar el año del Copyright de la web para el caso anterior).

En el repositorio de la asignatura se puede ver un ejemplo de esta estructura para separar el contenido de una página web. Es el ejemplo Ejercicio1 [https://bitbucket.org/sfaci/servidor-ejercicios/src/49544bcd62a3e3f3ea43fd7f0d8e8317ac5d34d1/Ejercicio1/?at=master] del repositorio servidor-ejercicios [http://bitbucket.org/sfaci/servidor-ejercicios]

Comentarios

- Podemos escribir comentarios dentro del código PHP
  - De una línea con `//` o `#`
  - De varias líneas con `/*` y `*/`
- Por supuesto, en la parte HTML podemos seguir incluyendo los comentarios con `<!-- -->`

Salida por pantalla

- La función echo() (aunque no es necesario usar los paréntesis) permite generar la salida del script PHP que será el código HTML que formará la página web resultante de procesar dicho script.
- El operador para concatenar más de un valor es el caracter `.`

Variables y tipos

- No hay que declarar las variables y el tipo de éstas se infiere al asignarle un valor. La variable tomará el tipo de ese valor. Trabajaremos principalmente con 4 tipos: **float**, **integer**, **string** y **boolean**. Además, también pueden aparecer estructuras compuestas como **arrays** de cualquier de los 4 tipos mencionados
- Una variable cambia su tipo automáticamente si se le asigna un nuevo valor de uno diferente

Ficheros HTML / PHP. Extensiones

Por otro lado, podemos tener código HTML para presentar información en el navegador. A estos ficheros los almacenaremos con la extensión *.html*. El navegador es capaz de interpretarlos puesto que 'entiende' el código HTML. Es por eso que además podemos abrirlos directamente con Firefox o Chrome directamente.

hola.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primera página web</title>
  </head>
  <body>
    <p>Esta es mi primera página web</p>
    <a href="#">Esto será un enlace</a>
    <img src="#" alt="Esto será una imagen"/>
  </body>
</html>
```

Pero hay que tener en cuenta que, a partir de ahora, podremos encontrarnos ficheros que contengan ambos lenguajes mezclados (PHP siempre aparecerá indicando su inicio y fin con las etiquetas correspondientes). En ese caso, siempre tendremos que almacenar el fichero con la extensión *.php* para indicarle a *Apache* que hay código (aunque sea una sola línea) que debe ser procesador por el intérprete de PHP.

A veces, por comodidad, y por no tener que modificar más adelante el nombre de ningún fichero, es posible almacenar código HTML bajo la extensión *.php* si luego tenemos pensado añadir dicho código. Apache puede pensar que hay código que ejecutar, se lo pasará al intérprete de PHP y, al no tener nada que hacer, devolverá el propio código HTML.

Por todo lo explicado anteriormente, estos ficheros (*.php*), no se pueden abrir directamente con ningún navegador. Éstos son incapaces de interpretar el código PHP. Hay que acceder directamente desde el propio servidor web (utilizando el navegador). Por ejemplo, al siguiente script PHP accederiamos a través de la siguiente URL `http://localhost/hola.php` `[http://localhost/hola.php]` si el servidor web se encontrara instalado en nuestro propio equipo.

mipagina.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primera página web</title>
  </head>
  <body>
    <p>Esta es mi primera página web</p>
    <a href="#">Esto será un enlace</a>
    <img src="#" alt="Esto será una imagen"/>
  </body>
</html>
```

En este caso tenemos código PHP escrito junto con HTML.

mipagina.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primera página web</title>
  </head>
  <body>
    <?php echo "<p>Esta es mi primera página web</p>"; ?>
```

```
        <?php echo "<p>Hoy es " . date("d.m.Y") . "</p>"; ?>
        <a href="#">Esto será un enlace</a>
        <img src="#" alt="Esto será una imagen"/>
      </body>
    </html>
```

Ejecución de un script PHP

Lo que tenemos que tener en cuenta es que nuestra página web HTML puede contener código PHP que deberá ser procesado por el intérprete. Si no lo contiene será Apache quién sirva la página directamente. En caso de que lo contenga, éste tendrá que pasar el fichero al intérprete, que será quien genere el código HTML resultante (por el camino quizás deba conectarse a una Base de Datos para recoger alguna información).

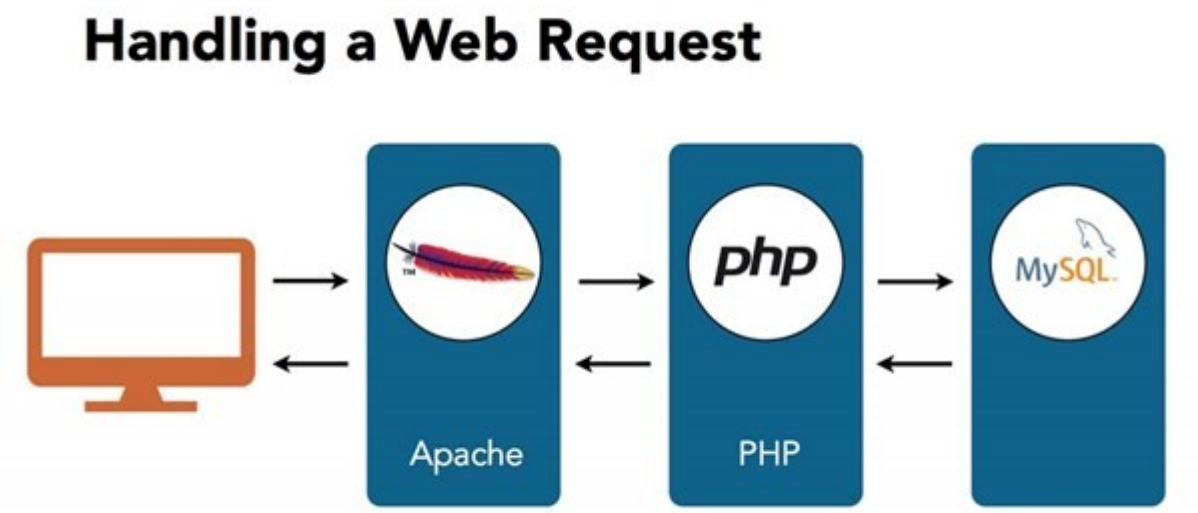


Figure 2: Ejecución de una página PHP

Variables y operadores

Sentencias de control

Sentencia if

```
if <condicion1> {
    <sentencias1>
}
[elseif <condicion2> {
    <sentencias2>
}]
. . .
[else {
    <sentencias3>
}]
```

```
<?php
$numero1 = 10;
$numero2 = 20;

if ($numero1 < $numero 2) {
    echo "numero1 es menor que numero2";
}
elseif ($numero1 > $numero2) {
    echo "numero1 es mayor que numero2";
}
```

```
else {
    echo "Los dos números son iguales";
}
```

Bucle For

```
for (<inicializacion>; <condicion>; <incremento>) {
    <sentencias>
}
```

```
<?php
$vector = array(1, 2, 3, 4);
for ($i = 0; i < len($vector); $i++) {
    echo $vector[i] + "<br/>";
}
```

Bucle foreach

```
foreach (<elemento> in <array>) {
    <sentencias>
}
```

```
<?php
$vector = array(1, 2, 3, 4);
foreach ($vector as $elemento) {
    echo $elemento . "<br/>";
}
```

Arrays y otras estructuras de datos

Declarar un array

Con la función `array()` podemos crear arrays de diferentes maneras, según se muestra a continuación:

```
// Declarar un array vacío
un_vector = array();
// Declarar un array con elementos
otro_vector = array("uno", "dos", "tres");
// Declarar un array asociativo
array_asociativo = array("DAW" => "Desarrollo de Aplicaciones Web",
                         "DAM" => "Desarrollo de Aplicaciones Multiplataforma");
```

Añadir elementos a un array

```
// Añade un elemento al final del array
otro_vector[] = "cuatro";
```

```
// Añade un elemento al final del array
array_push($otro_vector, "cinco");
// Añade un elemento al inicio del array
array_unshift($otro_vector, "cero");
```

Contar el número de elementos de un array

La función `count()` permite contar el número de elementos de un array

```
echo "Número de elementos " . count($otro_vector);
```

Ordenar un array

La función `sort()` ordena los elementos de un array de forma ascendente sobre el propio array.

```
sort($un_vector);
```

Paso variable de parámetros

Tanto en el lado de la definición de la función como en el momento del paso de parámetros, podemos usar una características de PHP que permite tanto definir funciones con número variable de parámetros como pasar un array como parámetro haciendo que PHP acepte realmente cada uno de sus elementos como los parámetros de la misma, aunque ésta haya sido definida con un número fijo de los mismos [1)].

En el primer caso podemos presentar la función indicando que acepta un número variable de parámetros. En este caso podemos pasarle tantos parámetros como queramos

```
<?php
function sumar(...$numeros) {
    $resultado = 0;
    foreach ($numeros as $numero) {
        $resultado += $numero;
    }
    return $resultado;
}

echo sumar(34, 45, 3, 2, 12);
?>
```

En el siguiente caso tenemos una función con un número fijo de parámetros y podemos pasar los elementos de un array con el operador ... para que cada elemento del mismo ocupe la posición de uno de los parámetros

```
<?php
function sumar($numero1, $numero2) {
    return $numero1 + $numero2;
}

$vector = array(10, 20);
echo sumar(...$vector);
?>
```

Ejercicios

Creación de funciones

funciones.php

```
      <?php
      function sumar($numero1, $numero) {
          $resultado = $numero1 + $numero2;
          return $resultado;
      }

<?php
include("funciones.php");

$resultado = sumar(3, 4);
echo "<p>El resultado es " . $resultado . "</p>";
```

Ejercicios

Acceso a Bases de Datos

Fichero de configuración

Lo primero que tendremos que tener en cuenta es que la configuración de acceso a la Base de Datos deberá estar en un fichero a parte y cada valor definido como una constante. De esta forma, cada vez que la queramos usar, este fichero podrá ser incluido para usar el valor de dichas constantes.

configuracion.php

```
<?php
define("DB_USUARIO", "santi");
define("DB_PASSWORD", "misupercontrasena");
define("DB_HOST", "localhost");
define("DB_NOMBRE", "nombrebasededatos");
```

Creación de Bases de Datos y/o tablas

Para el caso de webs que deban ser instaladas por los usuarios pueden resultar útil lanzar sentencias de creación de tablas, como el siguiente ejemplo donde un script PHP lanza la creación la Base de Datos, y a continuación la creación de una tabla.

```
<?php
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}
$sql = "CREATE DATABASE " . DB_NOMBRE;
$sentencia = $conexion->prepare($sql);
$sentencia->execute();

$conexion->select_db(DB_NOMBRE);

$sql = "CREATE TABLE usuarios (
        id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY
        nombre VARCHAR(255) NOT NULL,
        apellidos VARCHAR(255),
        email VARCHAR(255),
        telefono VARCHAR(50) UNIQUE)";

$sentencia = $conexion->prepare($sql);
$sentencia->execute();

$sentencia->close();
$conexion->close();
```

Inserción de registros

De forma similar podemos insertar registros en una Base de Datos utilizando sentencias SQL parametrizadas.

```
<?php
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}
$sql = "INSERT INTO usuarios (nombre, apellidos, email, telefono)
       VALUES (?, ?, ?, ?)";

$sentencia = $conexion->prepare($sql);
$sentencia->bind_param("ssss","Santiago", "Faci", "email", 123456789);
```

```
$sentencia->execute();

$sentencia->close();
$conexion->close();
```

Obtener el último id generado

En ocasiones será útil obtener el último *id* generado por una sentencia `INSERT` cuando tengamos que registrar otra información y relacionarla con la primera. No podemos conocer con qué *id* tengo que relacionar la información puesto que el *id* no lo conoceré hasta una vez introducido el dato. Por ejemplo, si tengo que registrar un pedido y todos sus detalles, primero insertaré el pedido pero necesitaré el *id* generado para ese pedido puesto que todos los detalles estarán relacionados con el pedido a través de dicho campo.

Hay una solución rápida pero incorrecta que sería lanzar una consulta del estilo a `SELECT MAX(id) FROM pedidos` para conocer el último *id* de esa tabla. La solución no es válida puesto que entre haber insertado el pedido y recoger el resultado de dicha consulta pueden haberse "colado" innumerables pedidos en la tabla (insertados por otros usuarios) y el valor que obtendría yo no sería el correcto.

Así, en PHP disponemos de un método, llamada `insert_id`, al que puedo invocar para obtener el último *id* generado por mi conexión a la Base de Datos, por lo que ya no podrán "colarse" los *ids* generados por otros usuarios. Veamos un ejemplo:

```
. . .
$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
. . .
$sql = "INSERT INTO pedidos (descripcion, precio, fecha_pedido)
       VALUES (?, ?, ?, ?)";

$sentencia = $conexion->prepare($sql);
$sentencia->bind_param("sds","Pedido del día", 120, "2017-11-21");
$sentencia->execute();

$id_pedido = $conexion->insert_id;
while ($detalles as $detalle) {
    $sql = "INSERT INTO detalles (id_pedido, id_articulo, precio) VALUES " .
           "(?, ?, ?)";
    $sentencia = $conexion->prepare($sql);
    $sentencia->bind_param("iid", $id_pedido, $detalle->id_articulo, $detalle->precio);
    $sentencia->execute();
}
. . .
```

Consulta de registros

En este primer ejemplo veremos como lanzar consultas SQL sobre la Base de Datos y recoger los resultados con un cursor (variable `$resultado`) para luego recorrer dicho cursor utilizando la función `fetch_row()` que devuelve una fila del resultado, en cada iteración del bucle donde se encuentra. De la fila (la variable `$fila`) podremos acceder a cada uno de los campos por la posición que ocupaban en la consulta (empezando con el valor `0`).

```
<?php
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}

$sql = "SELECT * FROM usuarios";
$sentencia = $conexion->prepare($sql);
$sentencia->execute();
$resultado = $sentencia->get_result();
while ($fila = $resultado->fetch_row()) {
    echo "<li>" . $fila[1] . "</li>";
}
$sentencia->close();
```

```
$conexion->close();
```

Consulta de registros utilizando un array asociativo

También podemos utilizar un array asociativo para almacenar el resultado de una consulta y de esa manera podremos acceder a cada campo de dicho resultado utilizando el nombre de la columna.

La forma de trabajar es la misma, solo que ahora obtendremos el valor de cada fila con la función `fetch_assoc()` que devuelve un array asociativo de forma que podemos obtener el valor de cada columna a partir del nombre de ésta con la variable `$fila` en este caso:

```
<?php
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}

$sql = "SELECT * FROM usuarios";
$sentencia = $conexion->prepare($sql);
$sentencia->execute();
$resultado = $sentencia->get_result();
while ($fila = $resultado->fetch_assoc()) {
    echo "<li>" . $fila["nombre"] . "</li>";
}
$sentencia->close();
$conexion->close();
```

Consulta de registros con parámetros

También puede ser necesario parametrizar nuestras consultas en base a un parámetro pasado por la URL, un dato introducido en un formulario, . . . de forma que podamos crear scripts más genéricos y polivalentes. En ese caso debemos usar las consultas parametrizadas de forma que una vez "preparada" la consulta podamos inyectarle nosotros los parámetros que se han indicado con el caracter `?`. Al asociaar los parámetros con sus valores, utilizando la llamada al método `bind_param` asociaremos cada parámetro con un valor, indicando ademas el tipo del parámetro con los siguientes caracteres:

- `s`: Si es una cadena (`string`)
- `d`: Si es un número con decimales (`double`)
- `i`: Si es un número entero o booleano (`integer`)

```
<?php
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}

$sql = "SELECT * FROM usuarios WHERE id = ?";
$sentencia = $conexion->prepare($sql);
$sentencia->bind_param("i",12);
$sentencia->execute();
$resultado = $sentencia->get_result();
while ($fila = $resultado->fetch_row(MYSQLI_ASSOC)) {
    echo "<li>" . $fila["nombre"] . "</li>";
}
$sentencia->close();
$conexion->close();
```

Crear una función para la consulta de registros

Y una vez vistas varias formas de lanzar consultas contra la Base de Datos, podemos aprovechar para ver alguna forma de

organizar el código de forma que no tengamos que escribir todas las instrucciones de conexión, preparación de la consulta, parámetros, etc cada vez que hagamos una consulta.

Por ejemplo, podríamos crear una función a la que pasando la sentencia escrita en lenguaje *SQL* y los parámetros (si los hubiera) y que fuera ésta la que conectara con la Base de Datos, preparara el paso de parámetros (en caso de que haya), lanzara la consulta y devolviera los resultados.

```
function lanzar_consulta($sql, $parametros = array()) {

    $conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
    $sentencia = $conexion->prepare($sql);
    if (!empty($parametros)) {
        $tipos = "";
        foreach ($parametros as $parametro) {
            if (gettype($parametro) == "string")
                $tipos = $tipos . "s";
            else
                $tipos = $tipos . "i";
            }
        $sentencia->bind_param($tipos, ...$parametros);
    }

    $sentencia->execute();
    $resultado = $sentencia->get_result();
    $sentencia->close();
    $conexion->close();

    return $resultado;
}
```

Así, cada vez que queramos lanzar una consulta sobre la Base de Datos simplemente tendríamos que hacer algo asi:

```
// Aqui utilizaríamos la función include() para incluir el fichero que contiene
// la declaración de la función
. . .
$sql = "SELECT nombre, apellidos FROM usuarios";
$resultado = lanzar_consulta($sql);
. . .
```

Y en el caso de que la consulta estuviera parametrizada:

```
// Aqui utilizaríamos la función include() para incluir el fichero que contiene
// la declaración de la función
. . .
$ciudad = . . .
$descuento = . . .
. . .
$sql = "SELECT nombre, apellidos FROM usuarios WHERE ciudad = ? AND descuento = ?";
$resultado = lanzar_consulta($sql, array($ciudad, $descuento);
. . .
```

Paginación de resultados

Cuando los resultados de una consulta generan una página demasiada extensa resulta bastante molesto por lo que se suele emplear la técnica de paginación para mostrar un número limitado de resultados por página incluso cuando hay muchos más. La solución pasa por generar una serie de enlaces a modo de páginas que permitan al usuario navegar entre los resultados de una manera más cómoda. También es posible incluso mostrar al usuario la posibilidad de ver todos los resultados en una sola página si él lo prefiere. En ocasiones esas páginas podrán ser solamente botones que permitan avanzar a la pagina anterior o a la página posterior y en otras ocasiones nos pueden mostrar un listado completo de todas las páginas (o no si éste es a su vez demasiado extenso) para que a su vez podamos avanzar más rápidamente saltándonos páginas intermedias. Esta es la solución que mostraremos aqui pero hay que tener en cuenta que no es la única como ya he comentado.

Lo primero que haremos será definir una constante que almacene el número de resultados que se mostrarán por página. Hay que tener en cuenta que este valor puede tener un valor por defecto pero también podemos dejar que sea el usuario quien lo

escoja.

```
<?php
. . .
define("TAMANO_PAGINA", 5);
. . .
```

Y ahora, a la hora de mostrar los resultados, hay que añadir el código necesario tanto para el cálculo del total de páginas necesarias para mostrar los resultados, como para mostrar sólamente los resultados que procedan. Además, habrá que dejar unos enlaces (o botones según se prefiera) que permitan al usuario navegar por las *páginas* de los resultados:

```
<?php
// Esta variable indica que página hay que cargar
if (isset($_REQUEST["pagina"]))
    $pagina = $_REQUEST["pagina"];
else
    $pagina = 0;

. . .
. . .
// Primero calculamos cuántos resultados hay
$sql = "SELECT COUNT(*) FROM articulos";
. . .
$total_articulos = . . .
// Calcula el número de páginas que hacen falta
$total_paginas = $total_articulos / TAMANO_PAGINA;
. . .
// Se traen sólo aquellas filas que se van a mostrar (según la página)
$sql = "SELECT nombre, descripcion FROM articulos LIMIT " .
        $pagina * TAMANO_PAGINA . " ," . TAMANO_PAGINA;
. . .
// Prepara los botones que paginan los resultados
for ($i = 0; $i <= $total_paginas; $i++) {
?>
    <a href="?pagina=<?= $i ?>"><?= $i + 1 ?></a>
<?php
}
?>
```

Modificación de registros

A la hora de modificar registros, tal y como hemos hecho en el caso anterior, utilizamos consultas parametrizadas donde nosotros mismos debemos luego inyectar el valor de los parámetros. En este caso, al tratarse de dos parámetros, una cadena ('s' de 'string') y un entero ('i' de 'integer'), lo indicamos como se puede ver en el siguiente fragmento de código:

```
<?php
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}
$sql = "UPDATE usuarios SET email = ? WHERE id = ?";

$sentencia = $conexion->prepare($sql);
$sentencia->bind_param("si","nuevo@email.com", 123);
$sentencia->execute();

$sentencia->close();
$conexion->close();
```

Eliminación de registros

De forma muy similar a los casos anteriores, podemos eliminar registros:

```
<?php
```

```
include("config/configuracion.php");

$conexion = new mysqli(DB_HOST, DB_USUARIO, DB_PASSWORD, DB_NOMBRE);
if ($conexion->connect_error) {
    die("La conexión ha fallado " . $conexion->connect_error);
}
$sql = "DELETE usuarios WHERE id = ?";

$sentencia = $conexion->prepare($sql);
$sentencia->bind_param("i", 123);
$sentencia->execute();

$sentencia->close();
$conexion->close();
```

Crear una clase para el acceso a la Base de Datos

Una vez conocidas las operaciones que podemos realizar sobre una Base de Datos (*MySQL* en este caso, aunque será muy similar para cualquier SGBD), a la hora de la verdad conviene organizar lo mejor posible el código de nuestra aplicación para que a la hora de acceder a la Base de Datos lo hagamos de la mejor manera posible. Por un lado desde el punto de vista del rendimiento y por otro lado desde el punto de vista de organizar y estructurar bien el código de nuestro proyecto. Es por eso que resulta muy recomendable desde este momento comenzar a utilizar el paradigma de Programación Orientado a Objetos [http://servidor.codeandcoke.com/doku.php?id=apuntes:paginas_web#programacion_orientada_a_objetos] con el que podemos contar en un lenguaje PHP para organizar, en este caso, todo el código que realiza accesos a la Base de Datos.

A continuación se muestra un ejemplo genérico de cómo podría organizarse una clase donde concentrar todo el código que realice acceso a Base de Datos de forma que pueda utilizarse desde cualquier documento PHP que necesite bien consulta, insertar, modificar o eliminar algún dato, o realizar cualquier otra operación en alguna tabla.

Db.php

```
<?php
/** Clase que permite comunicarse con la Base de Datos
*/
class Db {

    private $conexion;

    /**
     * Constructor. Permite conectar con la Base de Datos
     */
    function __construct() {
        $this->conectar();
    }

    /**
     * Conecta con la Base de Datos
     */
    function conectar() {
        // Suponemos que en un fichero de configuración hemos definido
        // unas constantes con la configuración de la conexión
        $this->conexion = new mysqli(HOST_BD, USUARIO_BD, CONTRASENA_BD, NOMBRE_BD);
    }

    /**
     * Desconecta de la Base de Datos
     */
    function desconectar() {
        $this->conexion->close();
    }

    /**
     * Obtiene el último generado en la última sentencia INSERT
     * \return El id que se generó
     */
    function ultimo_id() {
```

```
            return $this->conexion->insert_id;
        }

        /**
         * Lanza cualquier tipo de sentencia con o sin parámetros
         * \param sql La sentencia a ejecutar
         * \param parametros Los parámetros, si los hay
         * \return El resultado de la consulta
         */
        function lanzar_consulta($sql, $parametros = array()) {

            $sentencia = $this->conexion->prepare($sql);
            if (!empty($parametros)) {
                $tipos = "";
                foreach ($parametros as $parametro) {
                    if (gettype($parametro) == "string")
                        $tipos = $tipos . "s";
                    else
                        $tipos = $tipos . "i";
                }
                $sentencia->bind_param($tipos, ...$parametros);
            }

            $sentencia->execute();
            $resultado = $sentencia->get_result();
            $sentencia->close();

            return $resultado;
        }
    }
```

Ejercicios

1. Realiza una página web que muestra una lista de noticias (o extractos de noticias) de forma que sea posible eliminar cada una de ellas pulsando en un botón que aparecerá a la derecha de cada una de las mismas

---

Creación de formularios web y recuperación de la información

A la hora de programar un formulario, si queremos emplearlos para que nuestros usuarios suban ficheros al sitio web, lo primero que tendremos que tener en cuenta es la configuración en nuestro fichero `php.ini` y comprobar que está acorde con nuestras necesidades. En la siguiente tabla aparecen las tres directivas más importantes, para que sirven y qué valores pueden tomar.

directiva	Descripción	Valor
file_uploads	Indica si se permite o no la subida de ficheros	true / false
upload_max_filesize	Indica el tamaño máximo de un fichero subido	128M
post_max_size	Define el tamaño máximo de POST. Afecta, por tanto, a la subida de ficheros	128M

Table 1: Directivas de php.ini a tener en cuenta

Un formulario web tiene dos partes:

- Por un lado el código HTML (y la correspondiente CSS) que define el diseño y estructura del mismo

- Y por otra parte el código PHP (en nuestro caso) que se encarga de procesar la información y realizar las operaciones pertinentes

Así, supongamos un formulario para iniciar sesión en una página web. Por una parte tendríamos el código HTML que lo define:

```
<form action="procesar.php" method="post">
  <fieldset>
    <legend>Datos Personales</legend>
    Login: <br/>
    <input type="text" name="nombre" size="30"/><br/>
    Contraseña:<br/>
    <input type="password" name="contrasena" size="15"/><br/>
  </fieldset>
  <input type="submit" value="Enviar"/>
</form>
```

Y por otra parte, el correspondiente script PHP que lo procesa. En este caso, simplemente se muestran los dos campos que el usuario ha introducido como ejemplo de cómo recibir la información en el lado servidor utilizando la variable `$_REQUEST`

procesar.php

```
<?php
$nombre = $_REQUEST["nombre"];
$contrasena = $_REQUEST["contrasena"];

echo "Hola " . $nombre;
```

Cómo subir ficheros usando formularios

En el caso de los formularios para subir ficheros, hay que tener en cuenta que tendremos que definir el tipo de codificación del mismo para indicar que se adjuntarán archivos:

```
<!DOCTYPE html>
<html>
<body>

<form action="subir_fichero.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="fichero" id="fichero">
    <input type="submit" value="Upload Image" name="submit">
</form>
</body>
</html>
```

Y en el lado del servidor, habrá que recuperar el valor de los campos del formulario y el del archivos o archivos adjuntos, que además tendrán que ser copiados a la ruta destino que haya definida en la web para la subida de archivos:

```
<?php
if (isset($_FILES["fichero"])) {
  $directorio = "ficheros/";
  // Establece la ruta final del fichero (manteniendo su nombre original)
  $ruta_final = $directorio . basename($_FILES["fichero"]["name"]);
  // Ruta donde se encuentra temporalmente el fichero pendiente de la ubicación final
  $fichero_temporal = $_FILES["fichero"]["tmp_name"];

  // Mueve el fichero de su ubicación temporal a la ruta definitiva en el servidor
  move_uploaded_file($fichero_temporal, $ruta_final);
  echo "<p>El fichero se ha subido correctamente</p>";
}
?>
```

Si fuera necesario, también podemos obtener la siguiente información de un fichero que ha sido enviado a través de un formulario:

```
// Obtiene el tamaño del fichero en bytes
```

```
$tamano = $_FILES["fichero"]["size"];
// Obtiene el tipo del fichero (la extensión)
$tipo = $_FILES["fichero"]["type"];
```

Formularios con campos ocultos

También es posible enviar información oculta al usuario en un formulario. Es la forma que disponemos de enviar información fija independientemente de la que se introduzca en los campos. En ese caso el campo del formulario se define como `hidden` y se fija su valor en la definición del propio campo con el atributo `value`.

Más adelante en el script PHP podrá ser recuperado como un campo más.

```
<form action="procesar.php" method="post">
  <fieldset>
    <legend>Datos Personales</legend>
    Login: <br/>
    <input type="text" name="nombre" size="30"/><br/>
    Contraseña:<br/>
    <input type="password" name="contrasena" size="15"/><br/>
    <input type="hidden" name="accion" value="login"/>
  </fieldset>
  <input type="submit" value="Enviar"/>
</form>
```

Y al procesar dicho formulario:

procesar.php

```
<?php
$nombre = $_REQUEST["nombre"];
$contrasena = $_REQUEST["contrasena"];
$accion = $_REQUEST["accion"];

echo "Hola " . $nombre;
```

Envío de formularios con Ajax

En este caso vamos a utilizar jQuery [http://jquery.com] y un plugin para trabajar con formulario llamado jQuery Form Plugin [http://malsup.com/jquery/form/]. Con ellos podremos procesar formularios de una manera más cómoda.

Aqui tenemos un ejemplo de un formulario en el que el usuario debe introducir dos valores para dar de alta su usuario en una web. El formulario se ejecuta con Ajax utilizando la librería de *jQuery* y el plugin mencionado anteriormente por lo que, desde el punto de vista del desarrollador, el mensaje de respuesta del script PHP que procesa los datos (el `action`) nos viene devuelto y podemos pintarlo en la web simplemente utilizando la función `echo` en dicho script. Será *jQuery* quién recogerá el valor y nos lo trairá de vuelta al documento HTML donde se encuentra el formulario.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de formularios</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7/jquery.js"></script>
  <script src="http://malsup.github.com/jquery.form.js"></script>
  <script>
  $(document).ready(function() {
    $('#formulario').ajaxForm({
      success:function(response) {
        // Se puede emitir una alerta
        alert("El usuario ha sido dado de alta correctamente");
        // También se puede escribir la respuesta del script en una capa
        $('#mensaje').html(response);
        // Y resetear el formulario
        $('#formulario').each(function() {
          this.reset();
        });
      }
    })
  });
  </script>
</head>
<body>
<h1>Ejemplo de formulario</h1>
<form id="formulario" action="alta_usuario.php" method="post">
  <fieldset>
    <legend>Datos Personales</legend>
    Nombre: <br/>
    <input type="text" name="nombre" size="30"/><br/>
    E-mail:<br/>
    <input type="text" name="email" size="30"/><br/>
  </fieldset>
  <input type="submit" value="Enviar"/>
</form>
<div id="mensaje"></div>
</body>
</html>
```

Aqui podemos ver con más detalle el código javascript utilizado, donde se indica qué hacer en caso de que el formulario se ejecuta correctamente (`success`).

```
$(document).ready(function() {
  $('#formulario').ajaxForm({
    success:function(response) {
      // Se puede emitir una alerta
      alert("El usuario ha sido dado de alta correctamente");
      // También se puede escribir la respuesta del script en una capa
      $('#mensaje').html(response);
      // Y resetear el formulario
      $('#formulario').each(function() {
        this.reset();
      });
    }
  })
});
```

Y, finalmente, el script PHP que procesa el formulario simplemente necesita *pintar* en pantalla el resultado y será *jQuery* quién lo recoja y lo muestre donde hemos indicado utilizando javascript.

alta_usuario.php

```
<?php
$nombre = $_POST["nombre"];
$email = $_POST["email"];
. . .
// Aqui se da de alta al usuario, por ejemplo
```

```
. . .
// Si hay algún error podemos emitir un mensaje directamente
echo "Se ha producido un error";
. . .
// Si todo va bien podemos notificarlo al usuario
echo "El usuario " . $nombre . " ha sido dado de alta correctamente";
```

Validación de formularios con Ajax

A la hora de validar los campos de un formulario, podemos comprobar, utilizando de nuevo el plugin jQuery Form Plugin [http://malsup.com/jquery/form/], si los valores introducidos en los formularios son correctos antes de que éstos sean enviados. Es una buena forma de evitar tener que enviar de vuelta los mensajes de error validando los formularios en el lado servidor desde PHP. Ahora podemos validar el formulario antes de que sea enviado

Veamos un ejemplo que podríamos aplicar para el caso anterior.

```
$(document).ready(function() {
  $('#formulario').ajaxForm({
    beforeSubmit:function(data, form) {
      if (!form[0].email.value) {
        $('#mensaje').html("Debes indicar un email");
        return false;
      }
    }
  })
});
```

También podemos validar formularios utilizando jQuery Validation Plugin [https://jqueryvalidation.org/documentation/] de una forma muy sencilla. Basta con marcar las etiquetas `input` como `required`, asignarles `id` (para poder personalizar los mensajes) e invocar la función `validate()` del plugin

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de formularios</title>
  <script src="jquery-3.2.1.min.js"></script>
  <script src="jquery.validate.min.js"></script>
</head>
<body>
<h1>Ejemplo de formulario</h1>
<form id="formulario" action="alta_usuario.php" method="post">
  <fieldset>
    <legend>Datos Personales</legend>
    Nombre: <br/>
    <input type="text" name="nombre" size="30"/><br/>
    E-mail:<br/>
    <input type="text" name="email" size="30"/ required><br/>
  </fieldset>
  <input type="submit" value="Enviar"/>
</form>
<div id="mensaje"></div>
<script>
$("#formulario").validate({
  messages: {
    nombre: " Debes introducir tu nombre",
    email: " Debes introducir tu email"
  }
});
</script>
</body>
</html>
```

En la parte *javascript* se puede ver como se invoca el método `validate()` para lanzar la validación del formulario y como se han personalizado los mensajes de error para cada uno de los campos (utilizando el `id` de cada uno de ellos)

```
$("#formulario").validate({
  messages: {
    nombre: " Debes introducir tu nombre",
    email: " Debes introducir tu email"
  }
});
```

Ejercicios

1. Crea un formulario que sirva como conversor de monedas EURO-DOLAR
2. Crea un formulario que permita subir una imagen y visualizarla una vez termine. Añade control de errores para que sólo se permitan subir ficheros con extensiones *jpg* o *png*
3. Crea un formulario donde el usuario introduzca un nombre de usuario y contraseña. El formulario será procesado por un script PHP que buscará el usuario en una base de datos
4. Realiza un formulario que permite insertar información en una Base de Datos
5. Realiza una web que muestre información de una tabla
6. Añade lo necesario a la web anterior para que se pueda modificar y eliminar la información de esa tabla

---

Mantenimiento del estado. Cookies y Sesiones

El protocolo HTTP es el protocolo encargado de la transferencia de hipertexto, utilizado para la transferencia de los documentos HTML desde el servidor hasta el cliente. Puesto que se trata de un protocolo sin estado, la transferencia de cada documento se realiza de forma independiente, sin mantener ninguna relación con cualquier transferencia que se haga inmediatamente anterior o posteriormente. Por tanto, este protocolo es incapaz de saber si es el mismo cliente quién está solicitando una serie de documentos (páginas web) o se trata de clientes diferentes.

Es por eso que necesitaremos algún mecanismo si queremos ser capaces de mantener la sesión de un usuario o saber de alguna manera que éste ha realizado alguna visita previa a la actual. En este caso, mediante PHP como lenguaje de desarrollo del lado servidor, veremos cómo llevar a cabo este tipo de operaciones.

En el desarrollo web existen dos formas para mantener el estado:

- Cookie: Es la información que un servidor web almacena en un cliente con el objetivo de mantener un estado o recordar algo sobre éste (para su uso en futuras visitas).
- Sesión: También es información que se utiliza para mantener el estado pero en este caso ésta se almacena en el servidor

También existen algunas otras diferencias entre cookies y sesiones como por ejemplo el hecho de que las sesiones se eliminen al cerrar el navegador mientras que las cookies pueden permanecer durante mucho tiempo en el equipo del cliente (y a pesar de haber cerrado el navegador).

Configuración de PHP

Antes de comenzar con el uso de sesiones en nuestros proyectos PHP tendremos que comprobar que éste está correctamente configurado para el uso de las mismas. Para eso, tendremos que echar un vistazo al fichero de configuración `php.ini` y comprobar que está fijado un valor para la variable `session.save_path` (que por defecto será `/tmp`).

php.ini

```
. . .
session.save_path="/tmp"
. . .
```

Formulario y scripts para inicio de sesión en un sitio web

La forma más habitual de iniciar una sesión en un sitio web es hacerlo a través de un formulario que permita identificar al usuario que va a iniciarla. En nuestro ejemplo preparamos un formulario sencillo donde un visitante necesita introducir su

nombre de usuario y contraseña (previamente se habrá registrado) para comenzar la sesión en el sitio web.

Suponemos que una vez iniciada su sesión, tendrá acceso a una zona del sitio web que vendrá personalizada. En caso contrario no tendría sentido solicitar que se identificara.

```
<form action="login.php" method="post">
  <fieldset>
    <legend>Login</legend>
    Usuario: <br/>
    <input type="text" name="usuario" size="30"/><br/>
    Contraseña:<br/>
    <input type="password" name="contrasena" size="15"/><br/>
  </fieldset>
  <input type="submit" value="Entrar"/>
</form>
```

El script que inicia la sesión tendrá que hacerlo utilizando el método `session_start()`. En ese momento el servidor almacenará la información necesaria para mantenerla mientras el usuario no decida cerrarla (veremos como se hace más adelante).

Además podemos aprovechar para almacenar variables de sesión (se almacenan en `$_SESSION`) de forma que podamos también *recordar* cierta información que pueda ser útil durante la misma. Una vez iniciada la sesión y recogida la información podemos redirigir al usuario a la página de inicio de nuevo o bien a una zona privada.

login.php

```
<?php
session_start();
$_SESSION["usuario"] = $_POST["usuario"];
$_SESSION["fecha"] = date()

header("location: index.php");
```

Hay que tener en cuenta que todas las páginas web del sitio que necesiten conocer el estado de la sesión deberán incluir la llamada a `start_session` para mantenerla y además habrá que comprobar que se tiene acceso al menos a la variable de sesión que utilizamos para identificar al usuario. En nuestro caso, en el script que procesa el formulario de login habíamos almacenado el nombre del usuario en la variable de sesión `$_SESSION["usuario"]` por lo que comprobamos que ésta exista para poder asegurar que el visitante actual se encuentra identificado. En caso contrario lo redireccionaremos fuera de la zona privada.

Hay que tener en cuenta que almacenando el valor del usuario que tienen iniciada la sesión permite a su vez personalizar la información que se muestra o bien permitir hacer determinadas acciones en función de quién sea ese usuario o bien del rol que tenga en el sitio web (si lo hemos almacenado en otra variable de sesión o podemos ir a consultarlo a la Base de Datos a partir de conocer el nombre de usuario).

index.php

```
<?php
session_start();
if (!isset($_SESSION["usuario"])) {
    header("location: no_permitido.php");
}
?>
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>Bienvenido a tu sesión de usuario <?= $_SESSION["usuario"]; ?></p>
<p>Puedes pasarte por el sitio web <a href="otra_pagina.php">aqui</a></p>
<p>También puedes <a href="logout.php">salir</a></p>
</body>
</html>
```

Cualquier otra página que necesite mantener el estado deberá iniciar de la misma forma, manteniendo la sesión con

`session_start` y comprobando que el visitante la tiene iniciada.

otra_pagina.php

```
<?php
session_start();
if (!isset($_SESSION["usuario"])) {
    header("location: no_permitido.php");
}
?>
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<p>Hola de nuevo <?= $_SESSION["usuario"]; ?></p>
</body>
</html>
```

Y finalmente, el script que procese (bien a través de un botón de un formulario o bien un simple enlace) la salida de la sesión tendrá que iniciarla para luego eliminar toda la información de la sesión (`session_unset`) y luego eliminar la propia sesión del servidor (`session_destroy`).

A partir de este momento, el servidor web dejará de recordar al visitante como un usuario con sesión a todos los efectos.

logout.php

```
<?php
session_start();
session_unset();
session_destroy()

header("location:index.php");
```

Ejercicios

1. Rediseña el ejemplo de los apuntes para que el formulario de login aparezca en la página principal (*index*) pero sólo cuando el usuario no haya iniciado sesión.
2. Realiza los cambios necesarios en la estructura de la web de forma que no sea necesario repetir tanto código (control de la sesión, cabeceras de la web, . . .)
3. Realiza un pequeño sitio web donde un usuario pueda registrarse como usuario, iniciar sesión y modificar los datos de su perfil. Una vez termine podrá cerrar su sesión

---

Programación Orientada a Objetos

El lenguaje PHP también incorpora el paradigma de Orientación a Objetos y es posible crear clases de la que luego instanciar objetos de forma bastante similar a como se hace en lenguajes como Java.

A continuación una clase con 3 atributos de diferente visibilidad, un constructor y algunos métodos con el objetivo de conocer la sintaxis con este paradigma.

```
<?php
class Producto
{
    public $marca;
    protected $precio;
    private $ingredientes;

    function __construct($marca, $precio)
    {
        $this->marca = $marca;
        $this->precio = $precio;
```

```
        $this->ingredientes = array();
    }

    function anadir_ingrediente($ingrediente) {
        array_push($this->ingredientes, $ingrediente);
    }

    function numero_ingredientes() {
        return count($this->ingredientes);
    }
}

$colacao = new Producto("Nestle", 10);
echo $colacao->marca;
echo $colacao->precio;
echo $colacao->numero_ingredientes();
```

Ejercicios

1. Reescribe el ejercicio 3 del punto anterior utilizando Programación Orientada a Objetos

---

Seguridad

Configuración de PHP

El primer paso para implementar unas buenas medidas de seguridad es no mostrar información excesiva a potenciales atacantes. Por eso, desde el fichero de configuración de PHP (`php.ini`), podemos activar los *logs* de errores y desactivar el mostrarlos por pantalla. Mostrar los fallos cuando éstos se produzcan puede ayudar a los posibles atacantes a conocer mejor nuestro sistema, lo que facilitará que encuentres vulnerabilidades.

php.ini

```
. . .
log_errors = On
display_errors = Off  # En entorno de desarrollo/pruebas puede estar a On
. . .
```

Inyección SQL

La inyección SQL es un tipo de ataque que consiste en inyectar código SQL a las aplicaciones que trabajan con Base de Datos para conseguir más información de que la éstas proporcionan (por motivos de privilegios), volcar la Base de Datos, realizar cambios, autenticarse, entre otros. La inyección de SQL se realiza, por ejemplo, añadiendo cierto código SQL en los formularios que debemos cumplimentar para acceder a cierta información o al autenticarnos. Se trata de aprovechar despistes o fallos en la programación que permitan ejecutar dichas consultas libremente. Así, podremos realizar cualquier operación directamente sobre la Base de Datos, dependiendo de cómo esté implementada el sitio o aplicación web.

En este caso, vamos a tratar con un tipo de inyección SQL que nos va a permitir autenticarnos en un sitio web conociendo solamente el usuario.

Suponemos que tenemos una Base de Datos de usuarios como la que se muestra a continuación

```
mysql> select * from usuarios;
+----+---------+------------------------------------------+-------+
| id | usuario | password                                 | email |
+----+---------+------------------------------------------+-------+
|  1 | santi   | 9d4e1e23bd5b727046a9e3b4b7db57bd8d6ee684 | NULL  |
```

```
+----+---------+-----------------------------------------+-------+
1 row in set (0.00 sec)
```

Además, disponemos de un formulario para iniciar sesión y se nos solicita el usuario y contraseña:

inicio.php

```
<!DOCTYPE html>
<html>
<head>
    <title>Formulario de inicio de sesión</title>
</head>
<body>
<form action="db.php" method="post">
    <fieldset>
         <legend>Datos Personales</legend>
        Login: <br/>
        <input type="text" name="usuario" size="30"/><br/>
        Contraseña:<br/>
        <input type="password" name="password" size="15"/><br/>
    </fieldset>
    <input type="submit" value="Enviar"/>
</form>
</body>
</html>
```

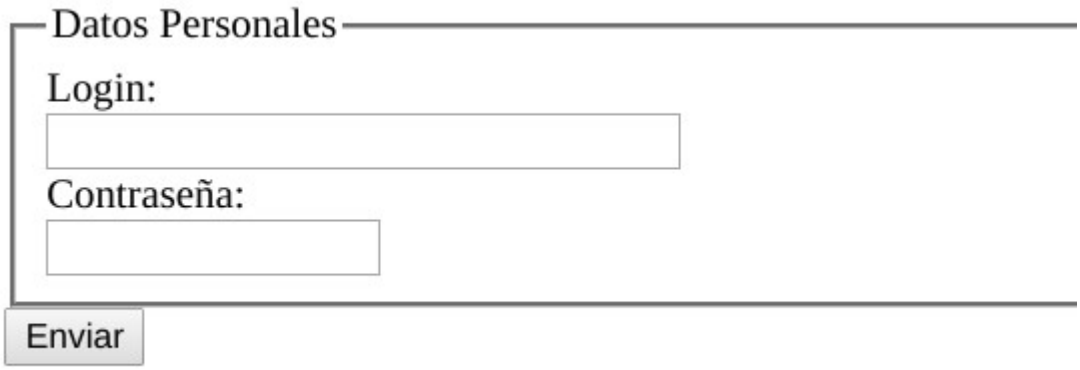


Figure 3: Formulario de inicio de sesión

Y aqui tenemos el script que se ha desarrollado para el inicio de sesión donde se validan usuario y contraseña y si coinciden, se muestra un mensaje confirmando que el usuario tiene acceso. Nos podemos fijar ya que los parametros de la consulta están simplemente concatenados, lo que es claramento un síntoma de script inyectable por SQL.

login.php

```
<?php
$usuario = $_POST["usuario"];
$password = sha1($_POST["password"]);

$conexion = new mysqli("localhost", "root", "", "programacionweb");
if ($conexion->connect_error) {
}

$sql = "SELECT * FROM usuarios WHERE usuario = '$usuario' AND password = '$password'";
echo "<p>$sql</p>";    # Sólo como depuración, para ver la inyección
$resultado = $conexion->query($sql);
if ($resultado->num_rows == 1) {
    echo "<p>Login OK!</p>";
}
else {
    echo "<p>Acceso no permitido</p>";
}
$conexion->close();
```

Ahora mismo nuestro script de login es vulnerable al menos a un tipo de ataque de inyección SQL muy extendido y reconocido. Si un usuario, que sólo conoce el nombre de usuario (pero no la contraseña), realiza este ataque, podrá iniciar sesión. Se trata simplemente de inyectar código SQL que será añadido al código que el propio script ejecuta. De esa forma podemos cambiar el comportamiento del script y trabajar de forma distinta para la que este diseñado.

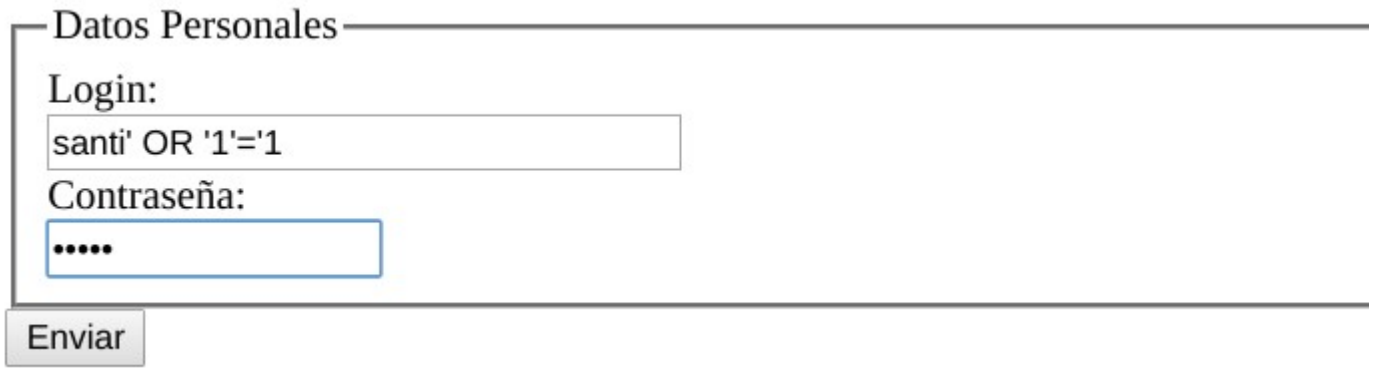


Figure 4: El usuario inyecta SQL en el formulario

Aqui se puede ver como el script muestra (a modo de depuración) la consulta SQL ejecutada y el resultado ha sido que se le ha permitido el acceso porque se ha validado dicha consulta como verdadera.

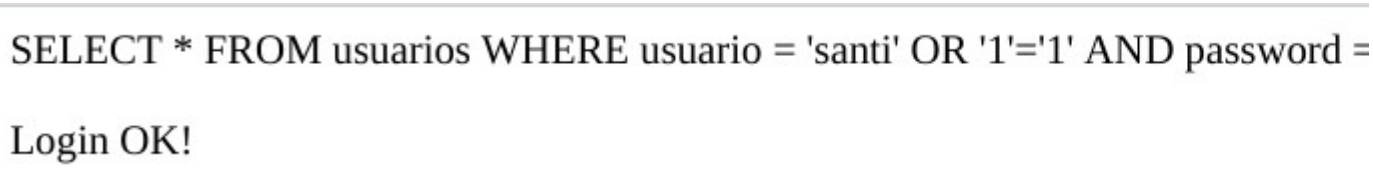


Figure 5: Inyección SQL realizada

Para subsanar este tipo de ataque, una de las principales medidas que se deben de realizar es parametrizar las consultas *SQL* utilizando los propios mecanismos que nos proporciona PHP.

Así, el script de *login*, parametrizando, en vez de concatenando los parámetros, deja de ser vulnerable para este ataque. Basta realizar unos pocos cambios tal y como se muestra en el siguiente ejemplo (este script reemplazaría al otro) y puede volver a probarse el formulario. Ahora no será posible realizar este tipo de inyección.

login2.php

```
<?php
$usuario = $_POST["usuario"];
$password = sha1($_POST["password"]);

$conexion = new mysqli("localhost", "root", "", "programacionweb");
if ($conexion->connect_error) {
}

$sql = "SELECT * FROM usuarios WHERE usuario = ? AND password = ?";
echo "<p>$sql</p>";

$sentencia = $conexion->prepare($sql);
$sentencia->bind_param("ss", $usuario, $password);
$sentencia->execute();
$sentencia->store_result();

if ($sentencia->num_rows == 1) {
    echo "<p>Login OK!</p>";
}
else {
    echo "<p>Acceso no permitido</p>";
}
$conexion->close();
```

Ejercicios

Creación de informes

Para la creación de informes en PHP existen multitud de librerías que disponen del API necesaria para la generación de documentos (PDF, por ejemplo) a partir de los datos que nosotros le proporcionemos (desde la Base de Datos, por ejemplo).

Una de esas librerías es FPDF [http://www.fpdf.org] que, aunque es muy sencilla, es muy recomendable para empezar a trabajar con informes en PHP. Para informes más completos donde queramos generar documentos con estilos más complejos podemos utilizar otras librerías como TCPDF [http://www.tcpdf.org].

Instalación de FPDF

Una vez descargado de su página web tendremos que copiar los ficheros necesarios (`fpdf.php`, `fpdf.css` y las carpetas `fonts` y `makefonts`) dentro de la carpeta donde guardemos las librerías que queremos usar en nuestro proyecto.

Creación de informes con FPDF

Veamos primero un ejemplo muy sencillo de la librería *FPDF* que nos va a permitir crear un documento PDF a partir de un texto. Conviene tener en cuenta que la llamada al método `Output()` genera el fichero PDF por lo que el siguiente código acaba lanzando el documento al navegador para cargarlo o descargarlo directamente.

```
<?php
require('fpdf.php');

$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Arial','B',16);
$pdf->Cell(40,10,'Mi primer informe');
$pdf->Output();
?>
```

Funciones internas

PHP dispone de numerosas funciones internas que se pueden usar directamente, sin necesidad de importar nada. Aqui se listarán, por categorías, las más utilizadas.

Funciones matemáticas

Nombre	Descripción
abs(numero)	Valor absoluto
ceil(numero)	Redondeo al alza
floor(numero)	Redondeo a la baja
is_nan(valor)	Comprueba si un valor es un numero

Nombre	Descripción
round(numero,precision)	Redondea con la precisión indicada
sqrt(numero)	Calcula la raíz cuadrada

Ejemplos:

```
$numero = abs(-4);          # $numero -> 4
$numero = ceil(3.1);        # $numero -> 4
$numero = floor(3.9);       # $numero -> 3
$es_numero = is_nan("hola") # $es_numero -> False
$numero = round(1.93847, 2) # $numero -> 1.94
$numero = sqrt(9)           # $numero -> 3
```

Funciones para cadenas de texto

Nombre	Descripción	Más info
implode([separador], array)	Une los elementos de un array en una cadena	implode() [http://php.net/manual/en/function.implode.php]
explode([separador], cadena)	Separa los elementos de una cadena en un array	explode() [http://php.net/manual/en/function.explode.php]
strlen(cadena)	Devuelve la longitud de una cadena	strlen() [http://php.net/manual/en/function.strlen.php]
substr(cadena, inicio, [longitud])	Devuelve parte de una cadena	substr() [http://php.net/manual/en/function.substr.php]
trim(cadena)	Elimina los espacios en blanco a ambos lados de una cadena	trim() [http://php.net/manual/en/function.trim.php]

Ejemplos:

```
$vector = array("uno", "dos", "tres");
$todos = implode(",", $vector);        # $todos -> 'uno,dos,tres'
$vector = explode(",", "uno,dos,tres") # $vector[0] -> 'uno' $vector[1] -> 'dos' . . .
$longitud = strlen("manzana");         # $longitud -> 7
$cadena = substr("murcielago", 2);     # $cadena -> 'rcielago'
$cadena = substr("murcielago", 2, 3);  # $cadena -> 'rci'
$cadena = trim("   manzana   ");       # $cadena -> 'manzana'
```

Funciones de fecha

Nombre	Descripción	Más info
date_create(fecha)	Crea una instancia de una fecha dada	date_create() [http://php.net/manual/en/function.date-create.php]
date_diff(fecha1, fecha2)	Devuelve la diferencia entre dos fechas	date_diff() [http://php.net/manual/en/function.date-diff.php]
date_format(fecha,formato)	Formatea una fecha según un patrón	date_format() [http://php.net/manual/en/function.date-format.php]
date(patron, [fecha=time()])	Devuelve una fecha (o la de hoy) formateada según un patrón	date() [http://php.net/manual/en/function.date.php]
strtotime(fecha)	Pasa una cadena a formato de fecha UNIX	strtotime() [http://php.net/manual/es/function.strtotime.php]

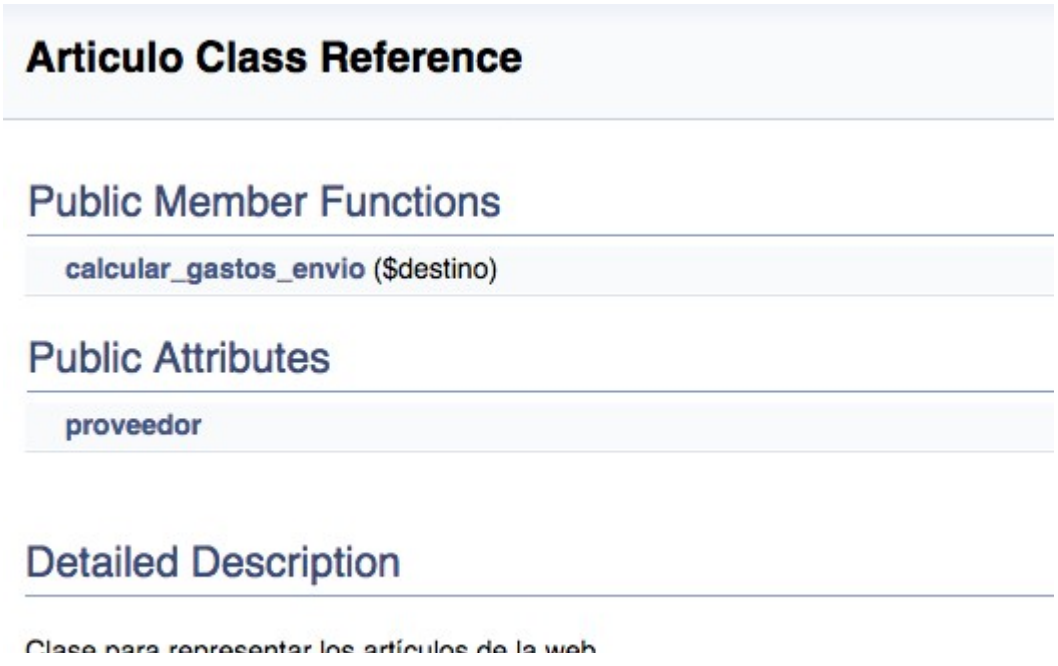
Ejemplos:

```
$fecha1 = date_create('2017-09-10');
$fecha2 = date_create('2017-09-17');
$diferencia = datediff(fecha1, fecha2) # $diferencia -> 7
$fecha = date_format(date(), "d-m-Y"); # $fecha -> '20-09-2017'
$fecha = date("d.m.Y");                # $fecha -> '20.09.2017'
// Cómo convertir una fecha extraída de un datepicker (jQuery) a formato MySQL
$fecha = date("Y-m-d", strtotime("20-10-2010")); # $fecha -> '2010-10-20'
```

Documentación

```
<?php
/**
 * Clase que representa a los artículos de la tienda
 */
class Articulo {

    /** Proveedor del artículo */
    public proveedor;
    . . .

    /**
     * Calcula los gastos de envio en función del destino
     * \param destino El destino del artículo
     * \return Los gastos de envío
     */
    function calcular_gastos_envio($destino) {
        . . .
        return gastos;
    }

    . . .
}
```

```
santi@zenbook:$ apt-get install doxygen graphviz
<code bash>

<code bash>
santi@zenbook:$ doxygen -g config.dox

. . .
INPUT =
. . .

santi@zenbook:$ doxygen config.dox
```



Articulo Class Reference

Public Member Functions

calcular_gastos_envio ($destino)

Public Attributes

proveedor

Detailed Description

Clase para representar los artículos de la web

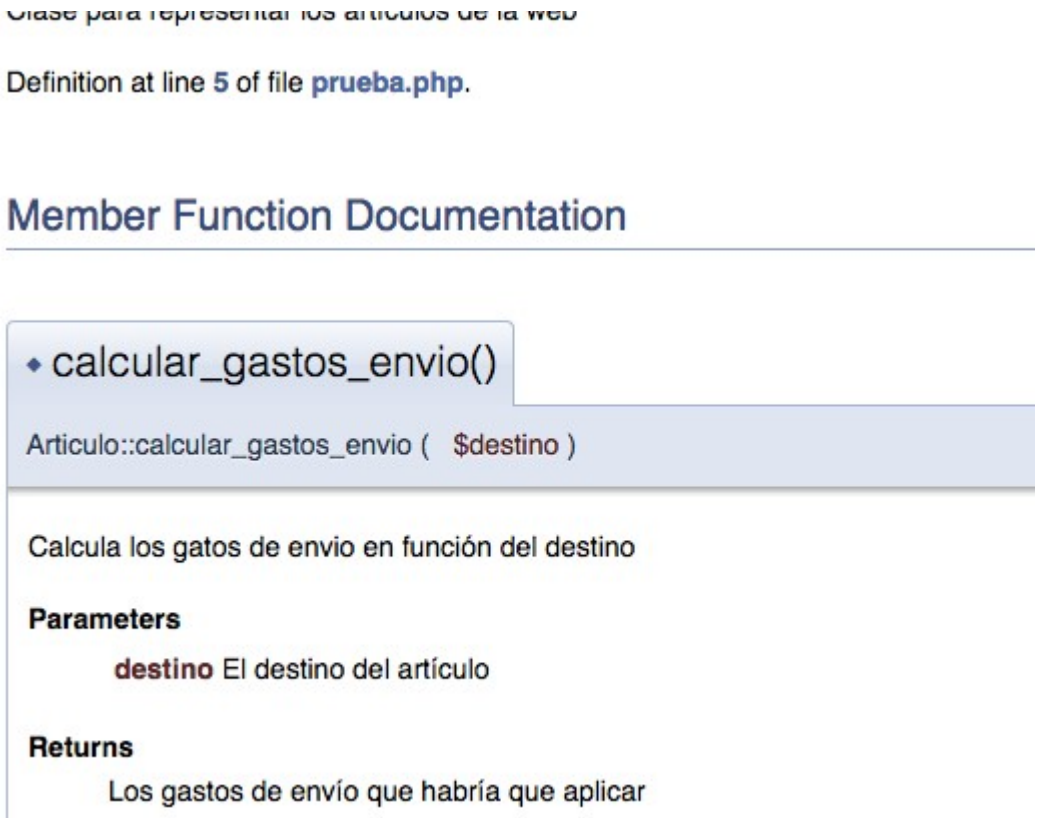
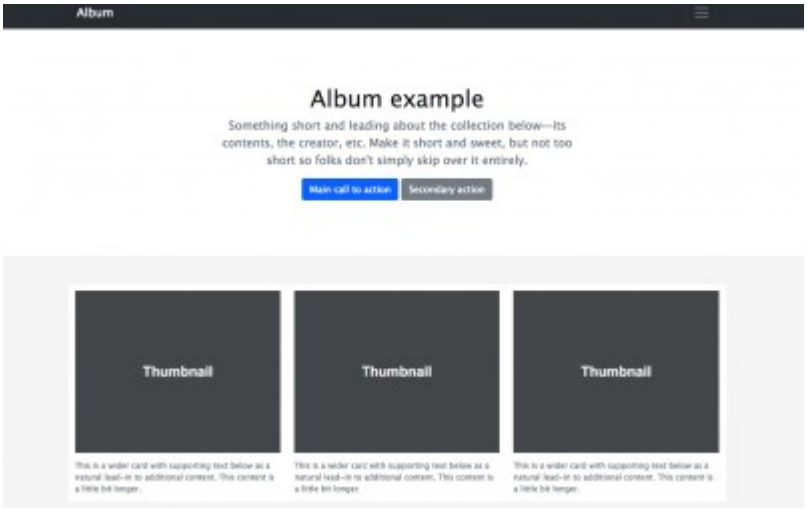


Figure 6: Clase documentada con Doxygen

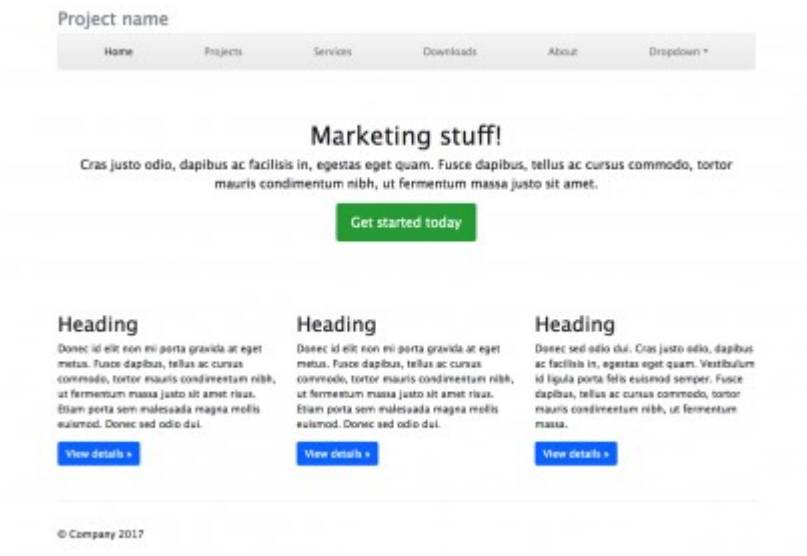
Ejercicios

- 1. Diseña un blog siguiente la plantilla Blog de Bootstrap [https://getbootstrap.com/docs/4.0/examples/blog/] para la parte pública y con la plantilla Dashboad de Bootstrap [https://getbootstrap.com/docs/4.0/examples/dashboard/] para la parte de administración. Tendrá las siguientes funcionalidades:
  - Ver las entradas del blog
  - Los usuarios tendrán que autenticarse para acceder al panel de administración
  - Escribir nuevas entradas
  - Crear, editar y eliminar usuarios del blog, que podrán tener los siguientes perfiles: 'Administrador', 'Editor' y 'Suscriptor'
    - Los Administradores podrán realizar cualquier tarea
    - Los Editores sólo podrán crear nuevas entradas y modificar o eliminar aquellas que hayan escrito ellos
    - Los Suscriptores sólo podrán acceder al panel de control para ver las entradas
  - En la barra lateral añade los siguientes componentes:
    - Un texto con un “Acerca de”
    - Listado de etiquetas utilizadas en las entradas y el número de entradas de cada una
  - Búsqueda de entradas

- 2. Realiza un sitio web para la gestión de un albúm de fotos siguiendo la plantilla Album [https://getbootstrap.com/docs/4.0/examples/album/] de Bootstrap. Puedes utilizar la plantilla Dashboad de Bootstrap [https://getbootstrap.com/docs/4.0/examples/dashboard/] para hacer el panel de control desde donde subir las fotos. El sitio web estaría pensado para mantener un solo albúm y que el usuario pueda configurar el título y la descripción que aparecen en la portada. Hay que tener en cuenta que quizás es necesario paginar los resultados.
  - Recuerda paginar [https://getbootstrap.com/docs/4.0/components/pagination/] los resultados cuando éstos sobrepasen una cantidad



- 3. Diseña ahora la web de gestión de albumes para que sea posible gestionar los albumes de varios usuarios de forma que cada uno de ellos, tras autenticarse, pueda actualizar solamente el suyo
  - En este caso, para los visiantes puedes utilizar la técnica de Migas de pan [https://getbootstrap.com/docs/4.0/components/breadcrumb/] para indicar al usuario dónde se encuentra en cada momento
- 4. Realiza un sitio web corporativo como imagen de una empresa siguiendo la plantilla Justified-nav [https://getbootstrap.com/docs/4.0/examples/justified-nav/] de forma que la empresa pueda gestionar a través de un pequeño panel de control las tres columnas que aparecen en portada
  - Modifica la portada para que en la parte central aparezca un Carrusel [https://getbootstrap.com/docs/4.0/components/carousel/] con las imágenes que el administrador del sitio configure



Proyectos de Ejemplo

En el repositorio de Github de servidor-ejercicios [https://github.com/codeandcoke/servidor-ejercicios] se pueden ir encontrando todos los ejemplos que vayamos haciendo en clase

---

1) http://www.php.net/manual/en/functions.arguments.php#functions.variable-arg-list [http://www.php.net/manual/en/functions.arguments.php#functions.variable-arg-list]