UD3 – Envío de datos y archivos al servidor

2º CFGS
Desarrollo de Aplicaciones Web

2024-25

1.- Peticiones HTTP

Las **peticiones HTTP son el medio** mediante el cual los **navegadores** (clientes web) **piden recursos a los servidores web**.

Las peticiones HTTP existentes son:

GET	HEAD	**POST**
DELETE	PUT	PATCH
TRACE	CONNECT	OPTIONS

Mediante las peticiones los **navegadores también pueden mandar información al servidor**.

1.- Peticiones HTTP

En esta parte del curso solo se usaran **GET** y **POST** que son las usadas en los enlaces y en los formularios HTML.

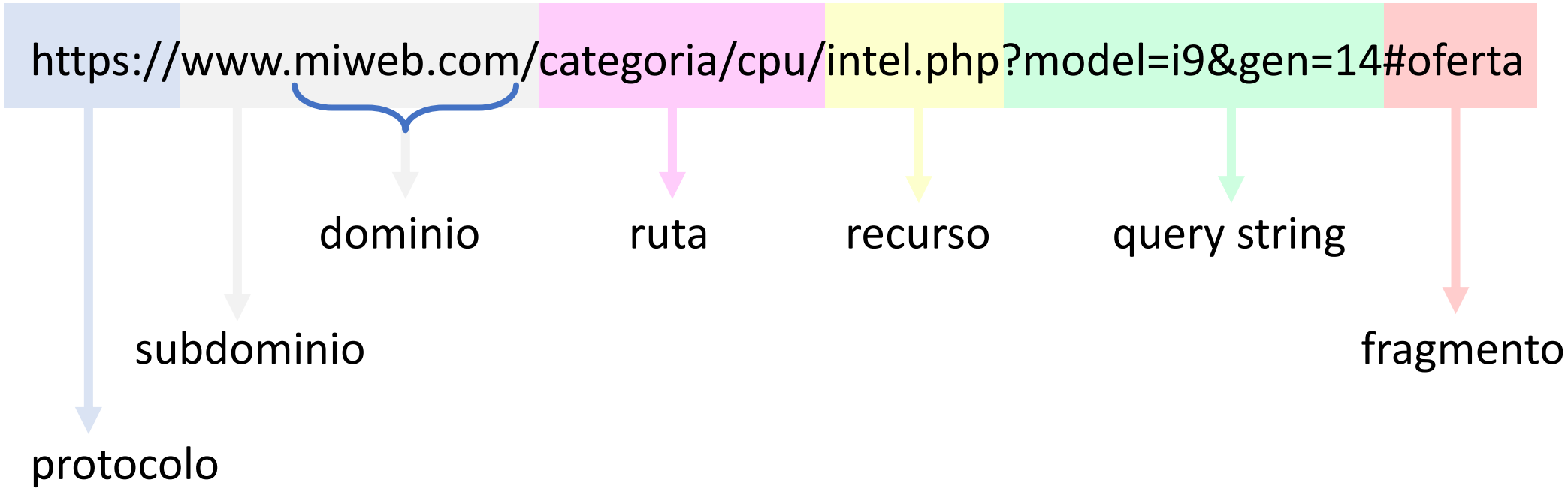
Más adelante con los servicios web API REST se verán los tipos PUT, PATCH y DELETE.

Con el estudio del framework Laravel en la unidad 9 se pondrán en práctica los tipos más comunes de peticiones: GET, POST, PUT, PATCH y DELETE.

1.- Peticiones HTTP

Las peticiones HTTP se realizan mediante la URL.

La estructura de una URL es la siguiente:



1.- Peticiones HTTP

El **navegador envía el tipo de petición** en las **cabeceras** de la petición HTTP.

Por **defecto** todas las peticiones que se realizan desde una aplicación web son peticiones del tipo **GET**.

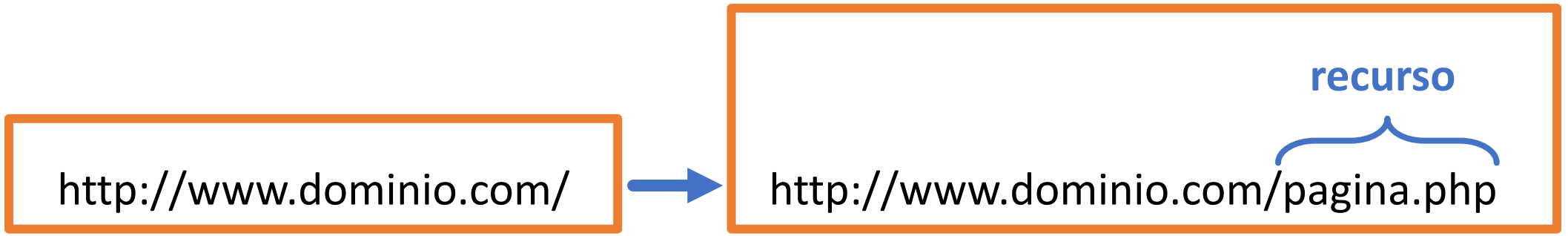
Cuando se envían datos mediante un formulario se puede elegir si la petición es de tipo **GET** o de tipo **POST**.

Lo más recomendable al enviar formularios es usar peticiones del tipo **POST**, aunque esto depende del tipo de datos enviado en el formulario:

- Datos **sin "importancia"**: como puede ser una búsqueda → GET
- Datos **sensibles**: como pueden ser los datos de login → POST

1.- Peticiones HTTP

- **GET**: cualquier petición realizada mediante un enlace (<a>).
  Se solicita un recurso específico del repositorio del servidor.



Se puede **enviar información** mediante la **query** de la URL.

1.- Peticiones HTTP

- **POST**: petición realizada desde un formulario.

  Se solicita un recurso específico del repositorio del servidor

  A la vez se **envían datos** para que un recurso específico del servidor los **procese**.

  Estos datos no se pueden ver en la URL.

  Se pueden añadir parámetros a la query string si fuera necesario.

2.- Variables superglobales

PHP dispone de un conjunto de variables que se pueden usar **en cualquier ámbito** sin necesidad de declararlas.

Estas variables son **arrays** que contienen información interesante del sistema: datos del servidor, datos del entorno, cookies…

$GLOBALS	$_SERVER	$_GET
$_POST	$_FILES	$_COOKIE
$_SESSION	$_REQUEST	$_ENV

Documentación oficial: variables superglobales.

2.- Variables superglobales

Gracias a las funciones **print_r** y **var_dump** se puede mostrar el contenido de una variable e información sobre su contenido.

Se puede incluir estas instrucciones en una etiqueta **<pre>** para visualizar mejor la información.

```
<pre>
    <?php
        print_r($_SERVER);
    ?>
</pre>
```

```
        Array
(
    [PATH] => C:\Program Files\PlasticSCM5\server;C:\Program Files
    [SYSTEMROOT] => C:/Windows
    [COMSPEC] => C:\Windows\system32\cmd.exe
    [PATHEXT] => .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;
    [WINDIR] => C:\Windows
    [SYSTEMDRIVE] => C:
    [TEMP] => C:/Windows/Temp
    [TMP] => C:/Windows/Temp
    [PHPRC] => C:/laragon/bin/php/php-8.3.11-nts-Win32-vs16-x64
    [PHP_FCGI_MAX_REQUESTS] => 0
    [_FCGI_SHUTDOWN_EVENT_] => 1180
    [SCRIPT_NAME] => /código apuntes/ud3/test.php
    [REQUEST_URI] => /c%c3%b3digo%20apuntes/ud3/test.php
    [QUERY_STRING] =>
    [REQUEST_METHOD] => GET
```

```
<pre>
    <?php
        var_dump($_SERVER);
    ?>
</pre>
```

```
        array(44) {
  ["PATH"]=>
  string(929) "C:\Program Files\PlasticSCM5\server;C:\Program File
  ["SYSTEMROOT"]=>
  string(10) "C:/Windows"
  ["COMSPEC"]=>
  string(27) "C:\Windows\system32\cmd.exe"
  ["PATHEXT"]=>
  string(62) ".COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MS
  ["WINDIR"]=>
  string(10) "C:\Windows"
  ["SYSTEMDRIVE"]=>
  string(2) "C:"
  ["TEMP"]=>
  string(15) "C:/Windows/Temp"
  ["TMP"]=>
```

2.- Variables superglobales

Variable	Información
$GLOBALS	Agrupa todas las variables globales.
$_SERVER	Información del servidor y del entorno de ejecución.
$_GET	**Datos en la query de la URL o de un formulario con método GET.**
$_POST	**Datos de un formulario con método POST.**
$_FILES	**Información de los archivos enviados con un formulario con método POST**
$_COOKIE	Información de las cookies de la aplicación web.
$_SESSION	Variables de sesión usadas por la aplicación.
$_REQUEST	**Toda la información de $_GET, $_POST y $_COOKIE junta.**
$_ENV	Variables del entorno de la aplicación web.

Documentación oficial: variables superglobales.

2.- Variables superglobales

Las variables superglobales **$_GET**, **$_POST** y **$_FILES** permiten acceder a la información recibida en las peticiones HTTP desde el script PHP solicitado en la petición.

Las variables $GLOBALS y $_REQUEST también incluyen la información de las variables anteriores, pero es recomendable utilizar la variable concreta del tipo de información recibida.

3.- Envío de datos mediante enlaces y query string

Mediante los **enlaces**, añadiendo **query string** se puede indicar al servidor que muestre una información determinada.

```html
<a href="index.php?user=alex">Ver publicaciones</a>
```

En el recurso destino se podrá acceder a las variables recibidas por query string mediante la variable superglobal **$_GET**.

```php
<?php
    echo 'Mostrando publicaciones de '. $_GET['user'];
    // Acceso a la base de datos para obtener las publicaciones del usuario
    // Recorrer los resultados y mostrarlos
```

4.- Envío de datos mediante formularios web

Los **formularios web** son la herramienta que **permite recoger datos introducidos** por el usuario y procesarlos en el servidor.

```
<form method="post" action="search.php">
    <!-- campos del formulario -->
</form>
```

Es importante el método (**method**) con el que se enviará el formulario al servidor (**get** o **post**).

En el atributo **action** se indica el recurso (**script PHP**) al que se enviarán los datos.

Para que un campo del formulario se envíe tiene que tener definido el atributo **name**.

En el recurso destino se podrá acceder a las variables recibidas desde el formulario mediante las variables superglobales **$_GET** o **$_POST** dependiendo del método usado.

4.- Envío de datos mediante formularios web

```
<form action="process.php" method="post">
    name del alumno: <input type="text" name="name" id="name">
    <br>
    Apellidos del alumno: <input type="text" name="surnames" id="surnames">
    <br>

    Ciclo que cursa:<br>
    <input type="radio" name="course" value="DAW"> Desarrollo de Aplicaciones Web
    <br>
    <input type="radio" name="course" value="DAM"> Desarrollo de Aplicaciones Multiplataforma
    <br><br>
    <input type="submit" value="Enviar">
</form>
```

4.- Envío de datos mediante formularios web

Archivo **process.php**:

```php
<?php
    echo 'El alumno ';
    echo $_POST['name'] .' '. $_POST['surname'];
    echo '<br>Se encuentra cursando el ciclo: ';
    echo $_POST['course'];
?>
```

Otra manera de mostrar la información:

```php
El alumno <?=$_POST['name']?> <?=$_POST['surname']?>
<br>
Se encuentra cursando el ciclo: <?=$_POST['course']?>
```

4.- Envío de datos mediante formularios web

En el caso de que un **checkbox** pueda enviar varios valores hay que indicar en el atributo **name** que es un **array**.

```
<form action="process.php" method="post">
    Nombre del alumno: <input type="text" name="name">
    <br>
    Módulos que cursa:<br>
    <input type="checkbox" name="subjects[]" value="DWES">
    Desarrollo Web en Entorno Servidor
    <br>
    <input type="checkbox" name="subjects[]" value="DWEC">
    Desarrollo Web en Entorno Cliente
    <br><br>
    <input type="submit" value="Enviar">
</form>
```

4.- Envío de datos mediante formularios web

Mediante el uso de arrays se puede organizar los datos enviados.

Aunque esta acción no se suele utilizar habitualmente.

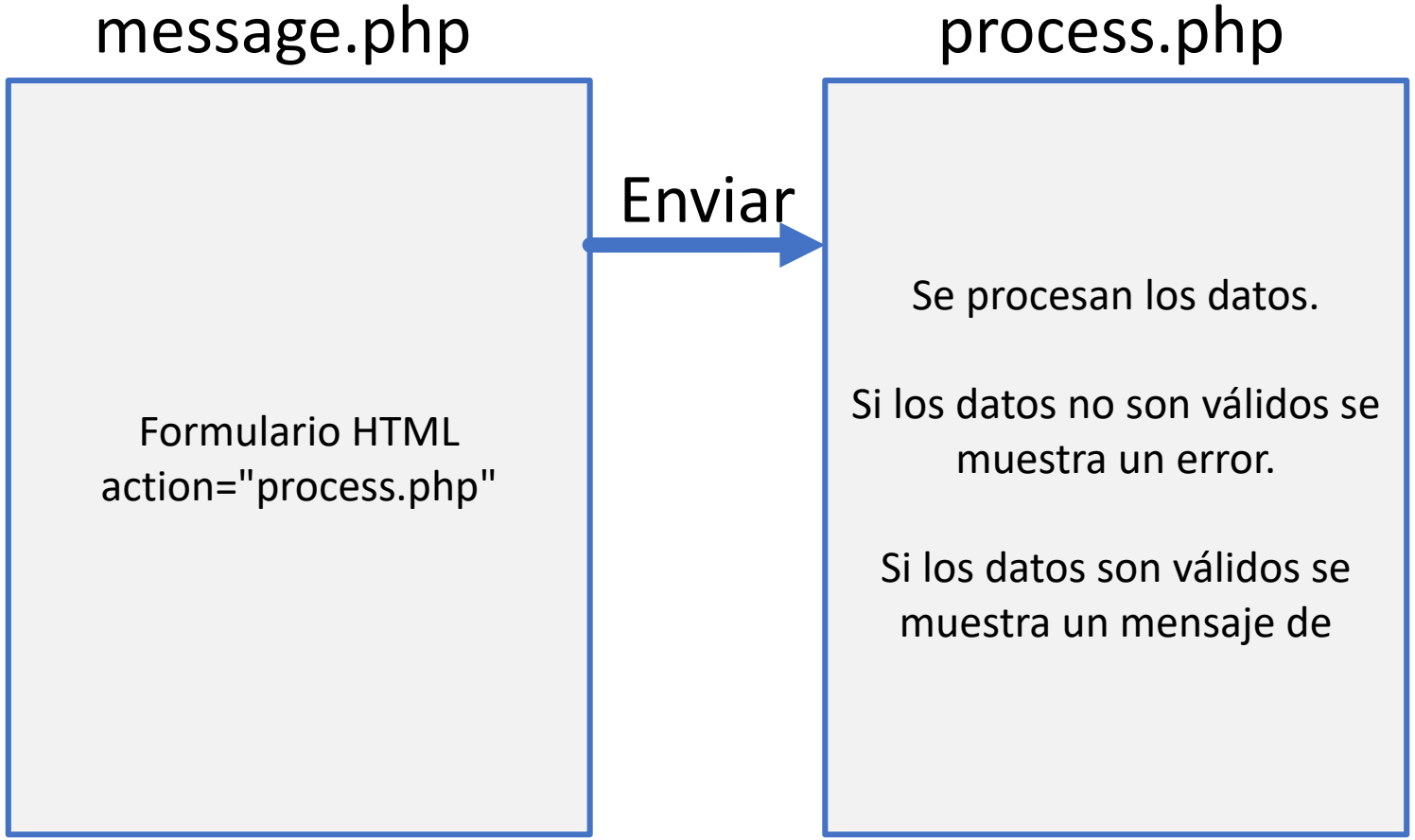
```
<form action="process.php" method="post">
    Jugador 1:<br>
    Nombre: <input type="text" name="player1[name]"><br>
    Apellidos: <input type="text" name="player1[surnames]"><br><br>
    Jugador 2:<br>
    Nombre: <input type="text" name="player2[name]"><br>
    Apellidos: <input type="text" name="player2[surnames]">
    <br><br>
    <input type="submit">
</form>
```

4.- Envío de datos mediante formularios web

A continuación, se muestran diferentes soluciones al flujo de datos del envío de datos al servidor mediante formularios.

4.- Envío de datos mediante formularios web

Caso 1



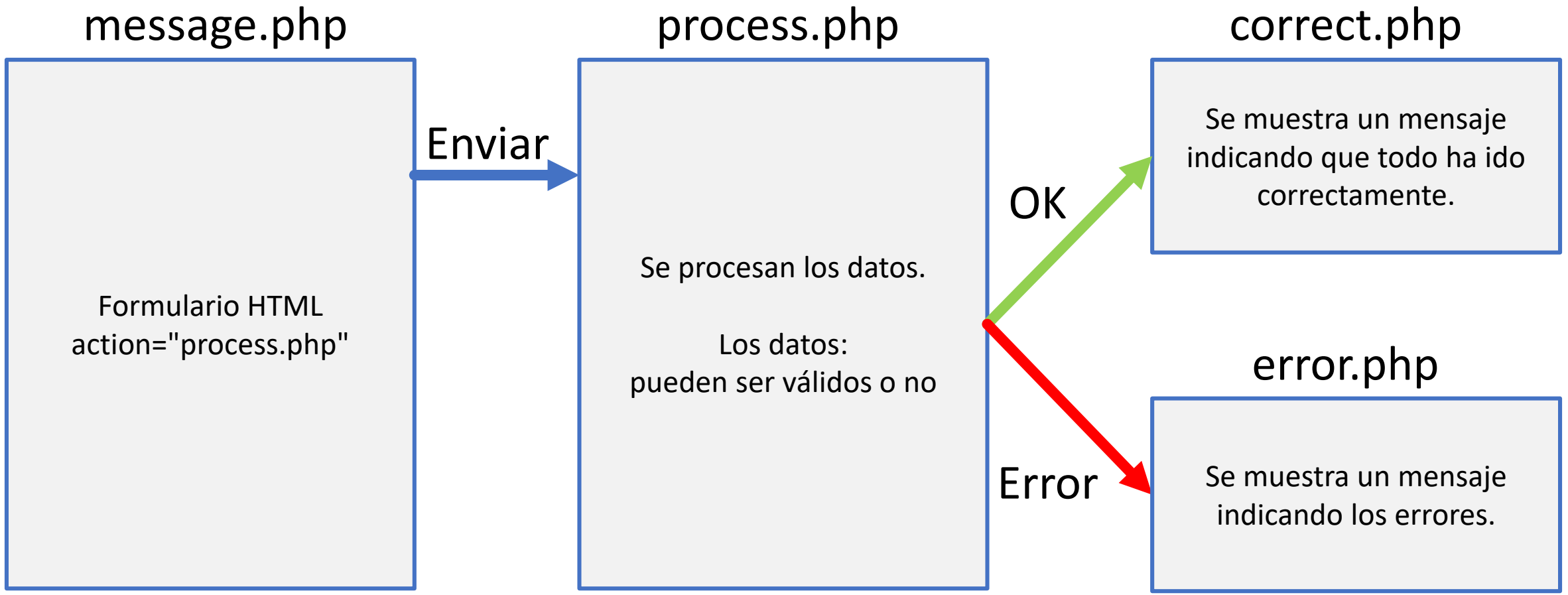
Práctica

Actividad 1:

Introduciendo datos en el almacén.

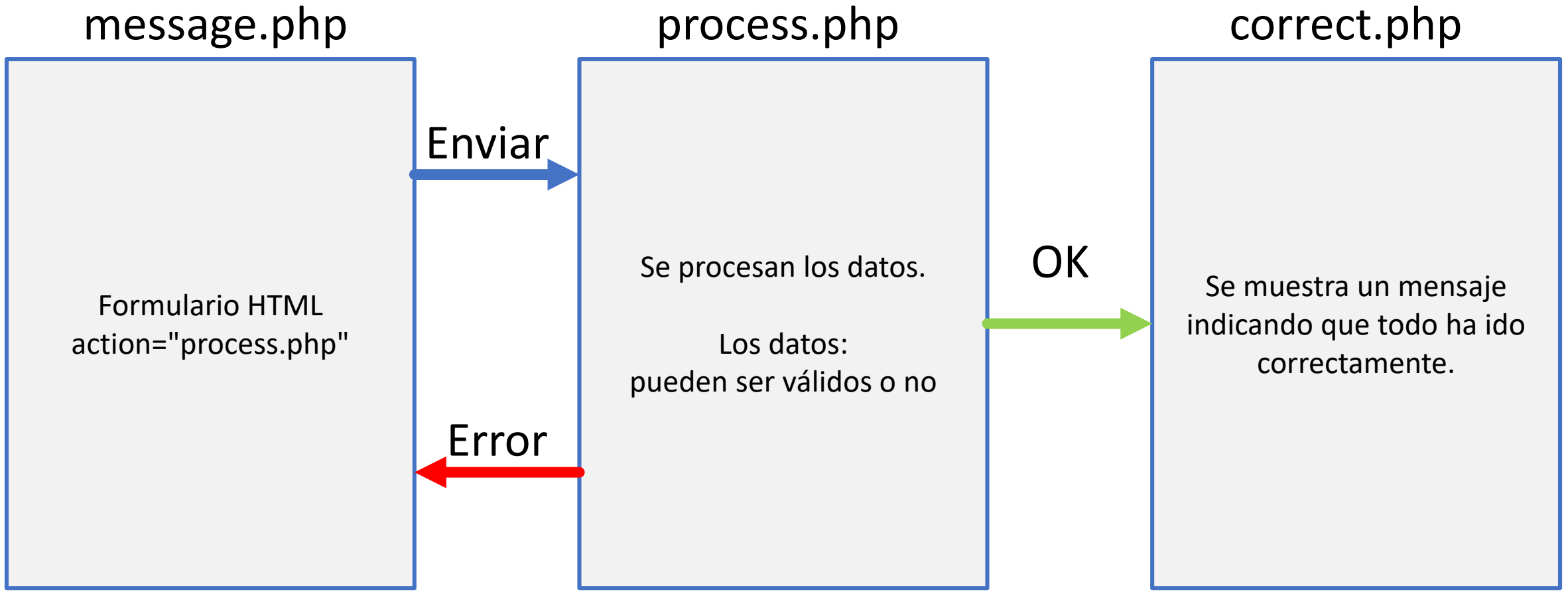
4.- Envío de datos mediante formularios web

Caso 2



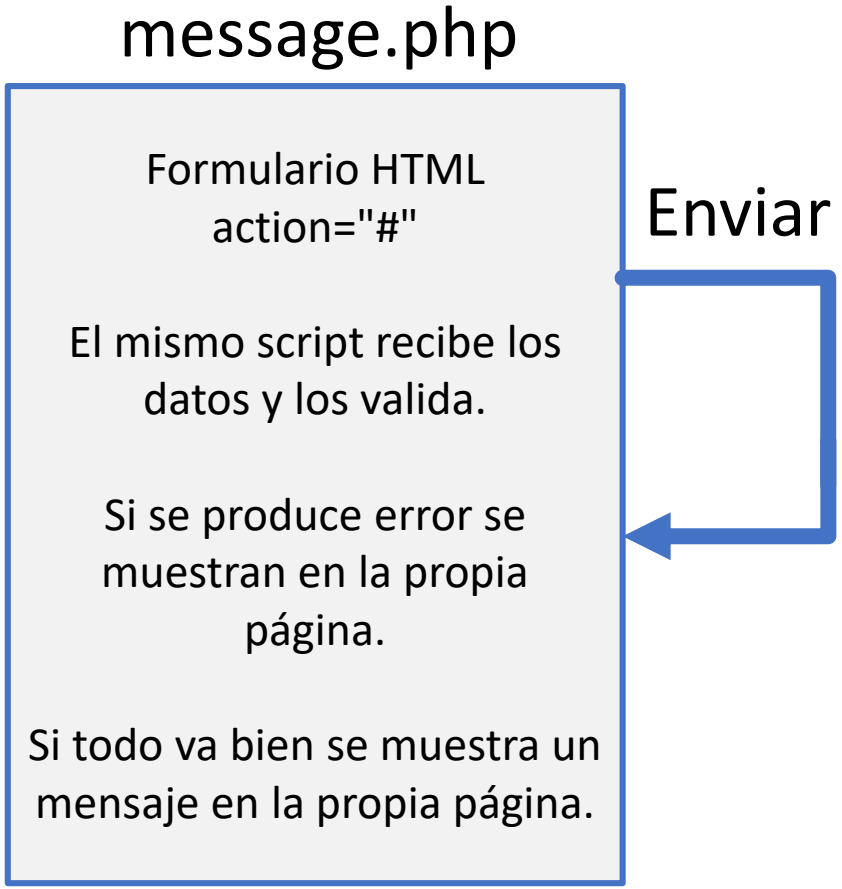
4.- Envío de datos mediante formularios web

Caso 3



4.- Envío de datos mediante formularios web

Caso 4 → El más utilizado



5.- Validación de datos

Cuando el usuario introduce datos en una aplicación web es importante **realizar la validación** de estos datos en todos los puntos vulnerables.

La validación se puede hacer en cuatro momentos:

Navegador

Cliente

Servidor

Además, en la base de datos también hay restricciones de tipos de dato.

5.- Validación de datos

- **Navegador**: mediante el uso de los tipos correctos en los campos **input** y el atributo **required** En las etiquetas **HTML**.

- **Cliente**: antes de ser enviados al servidor se validan en el cliente mediante **JavaScript**.

- **Servidor**: en el script que se reciben los datos se debe validar que estos son correctos mediante el lenguaje de programación del servidor (en clase PHP).
  También hay que evitar la suplantación de identidad y ataques **CSRF**. Para ello se hace uso de **input type="hidden"** y variables de sesión (esto se verá más adelante).

En una **aplicación web en producción** (accesible al público) **se deben validar los datos en todos los lugares** porque se puede alterar el comportamiento del navegador y de esa manera saltar la validación. ¿Cómo?

5.- Validación de datos

En las **actividades de clase:**

solo se validará en el servidor

para facilitar las comprobaciones y correcciones durante el desarrollo de las aplicaciones

5.- Validación de datos

Al recibir datos por la URL mediante la query string, existe la posibilidad de sufrir ataques ya que cualquier usuario puede alterar la URL.

Se espera una variable que debe ser numérica y se recibe en su lugar una cadena

http://midominio.com/process.php?query=vulnerable

Se espera una variable de longitud 4 y se recibe en su lugar una de otra longitud

http://midominio.com/process.php?id=26

En estos casos tras comprobar el dato se debe redirigir la petición y finalizar el script.

```
header('Location: /registro.php');
exit;
```

5.- Validación de datos

Para realizar todos los tipos de validaciones se utilizan funciones como **isset**, **is_numeric**, **strlen**, **strcmp**… y/o con **expresiones regulares**.

Se deben validar todos los campos recibidos y posteriormente informar de todos los errores que existan para así solo recargar la página web una vez.

Cuando se detecta algún error en algún campo se debe informar al usuario.

También se deberían rellenar los campos del formulario con los datos introducidos por el usuario para que este pueda comprobar dónde está el error y no obligar al usuario a volver a escribir los datos en el formulario.

5.- Validación de datos

El script PHP que recibe los datos del formulario puede ser el mismo que muestra el formulario:

action="#"

Así el comportamiento del script PHP con el formulario sería el siguiente:

- Si el script PHP no recibe variables desde el formulario se mostrará el formulario vacío.

- Si el script PHP recibe variables desde el formulario se procede a validar los datos.
  - Si se detectan vulnerabilidades o ataques se redirige al propio script PHP para cargarlo sin que reciba datos.
  - Si hay error en los datos se muestra el formulario con los datos introducidos por el usuario y además, se muestren los errores en los datos.
  - Si todos los datos son correctos se realiza su procesamiento, por ejemplo, guardar en la base de datos.

De esta manera se da más información al usuario sobre los problemas que se han producido.

5.- Validación de datos

Esquema de ejemplo de un script PHP con formulario que recibe los datos de su propio formulario.

Se comprueba si llegan datos mediante PHP
- Si hay ataques se redirige
- Si llegan datos se validan
- Si hay errores en los datos se guardan para mostrarlos posteriormente
- Si no hay errores se procesan los datos y se redirige a otro script o se tiene en cuenta para lo que se muestre en el cuerpo HTML*.

Se abre el cuerpo HTML
- Se muestra el formulario (o se muestra otro contenido*)
- Si hay errores guardados se muestran (todos juntos o al lado del campo del formulario al que pertenece el error).

Se cierra el cuerpo HTML

Práctica

Actividad 2:

Introduciendo datos en el almacén v2.0.

6.- Expresiones regulares

Las expresiones regulares son **patrones** de caracteres que permiten **comprobar** si **una cadena de caracteres** se ajusta al patrón o no.

Ejemplos típicos de comprobaciones con expresiones regulares:

- Que un nombre no tenga números.
- Que tenga una longitud concreta.
- Que se siga un orden en los caracteres (DNI → 8 números y 1 letra).
- Que sea un email.
- …

Las expresiones regulares existen en muchos lenguajes de programación así que es importante conocer su funcionamiento.

6.- Expresiones regulares

En PHP las expresiones regulares se escriben como cadenas de caracteres y van delimitadas al inicio y al final por el carácter: **/**

'/expresión_regular/'

El diseño de expresiones regulares es una disciplina cuyo nivel de dificultad aumenta conforme se quieren comparar patrones más complejos.

A continuación, se verá una pequeña guía del uso de las expresiones regulares que servirá para aprender a utilizarlas de una manera básica.

6.- Expresiones regulares

Modificadores

- **i**: no distingue entre mayúsculas y minúsculas (case-insensitive).

  '/a/i' → contiene la a o la A.

- **m**: busca el patrón en todas las líneas de la cadena, la cadena debe tener salto de línea: \n

- **u**: incluye caracteres UTF-16.

- **g**: no se detiene tras la primera coincidencia.

6.- Expresiones regulares

Agrupación

- **[ ]** [abc] contiene cualquier carácter de entre los indicados.
- **[^]** [^abc] contiene cualquier carácter que no sea de los indicados.
- **[-]** [0-9] contiene cualquier carácter que se encuentre en el rango.
- **[^ -]** [^A-B] contiene cualquier carácter que no esté en el rango.
- **(|)** (x|y) contiene uno de los caracteres (separador |).

Cantidad de caracteres

- **{ }** a{3} contiene exactamente 3 'a' seguidas.
- **{ ,}** a{3,} contiene 3 o más 'a' seguidas.
- **{ , }** a{3,5} contiene 3, 4 o 5 'a' seguidas.
- ***** a* contiene 0 o más 'a'. Similar: a{0, }
- **+** a+ contiene 1 o más 'a'. Similar: a{1, }
- **?** a? contiene 0 o 1 'a'. Similar: a{0,1}

6.- Expresiones regulares

Inicio – fin

- **^** ^hola empieza con "hola".
- **$** hola$ acaba con "hola".
- **^ $** ^hola$ exactamente "hola".

Otros

- **\s** el carácter espacio en blanco.
- **\S** cualquier carácter que no sea espacio en blanco.
- **\w** una letra.
- **\C** no es una letra.
- **\d** un dígito. Similar: [0-9]
- **\D** no es un dígito. Similar: [^0-9]

6.- Expresiones regulares

Ejemplos de expresiones regulares en PHP:

```php
// tiene 4 minúsculas
'/[a-z]{4}/'

// tiene 8 caracteres: letras y/o números
'/[a-zA-Z0-9]{8}/'

// 7 u 8 dígitos seguidos de 1 letra (DNI)
'/^\d{7,8}\w{1}$/'

// conjunto de letras, seguidas de arroba seguida, seguidas de un
// conjunto de letras, seguidas de un punto y seguido de 2 o 3
// letras → patrón simple para un email: rick_sanchez@mail.com
'/^[a-z_.]+@[a-z]+\.[a-z]{2,3}$/'
```

6.- Expresiones regulares

Ejemplos de expresiones regulares en PHP:

```
// conjunto de letras, seguidas de arroba seguida, seguidas de un
// conjunto de letras, seguidas de un punto y seguido de 2 o 3
// letras → patrón simple para un email: rick_sanchez@mail.com
'/^[a-z_.]+@[a-z]+\.[a-z]{2,3}$/'
```

Las expresiones regulares se pueden almacenar en variables.

```
$expr4Minusculas = '/[a-z]{4}/';
```

6.- Expresiones regulares

Si se quiere comprobar si una cadena cumple el patrón establecido por una expresión regular se usa la función preg_match.

```
preg_match(
    string $pattern,
    string $subject,
    array &$matches = ?,
    int $flags = 0,
    int $offset = 0
): int
```

Aunque esta función puede recibir hasta 5 parámetros bastará con utilizar los dos primeros para realizar las comprobaciones.

El valor devuelto será:

- **1** si la cadena cumple el patrón
- **0** si la cadena **no cumple** el patrón
- **false** si se produce un **error**.

6.- Expresiones regulares

Ejemplo de uso de **preg_match()**

```php
$exprPass = '/[a-zA-Z0-9]{8}/';
$pass = 'Xje3T8Uk';
if (!preg_match($exprPass, $pass)) {
    echo 'La contraseña debe tener al menos 8 letras y/o números';
}
```

Si se comprueba una variable recibida desde un formulario se usa $_POST

```php
$exprPass = '/[a-zA-Z0-9]{8}/';
$pass = 'Xje3T8Uk';
if (!preg_match($exprPass, $_POST['pass'])) {
    echo 'La contraseña debe tener al menos 8 letras y/o números';
}
```

6.- Expresiones regulares

La sintaxis para definir expresiones regulares en PHP se puede encontrar en la documentación oficial:

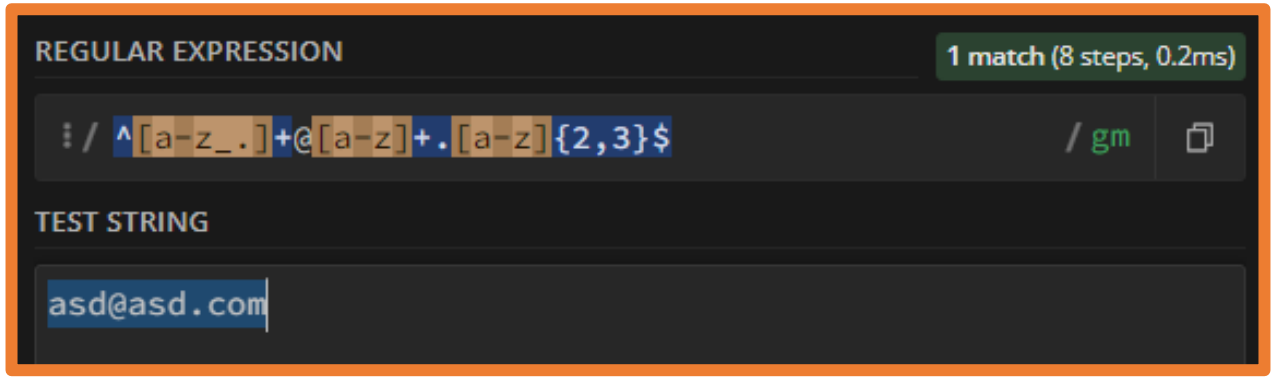
https://www.php.net/manual/en/reference.pcre.pattern.syntax.php

También se pueden encontrar Cheat Shhets:

https://quickref.me/regex

Existen herramientas para comprobar el funcionamiento de una expresión regular:

https://regex101.com/



Práctica

Actividad 3:

Validando la información.

Actividad 4:

Oferta de trabajo.

7.- Tipos MIME

MIME → **M**ultipurpose **I**nternet **M**ail **E**xtensions

En los inicios de internet, para enviar contenido por email se definió el **estándar MIME**.

Este estándar indicaba el tipo de contenido que se enviaba por email.

Hoy en día los tipos MIME se utilizan para cualquier envío de archivos por internet.

Cuando el servidor envía un archivo (html, php, png...) lo primero que hace es enviar las cabeceras en las cuales se indica el tipo MIME.

7.- Tipos MIME

El tipo MIME se indica mediante una cadena que tiene la siguiente estructura: tipo/subtipo

Algunos ejemplos:

Extensión	Tipo	MIME
.css	Hoja de estilo	text/css
.epub	Libro electrónico	application/epub+zip
.js	JavaScript	application/javascript
.json	Formato JSON	application/json
.png	Imagen PNG	Image/png

Lista completa de tipos MIME.

7.- Tipos MIME

Gracias a los tipos MIME los servidores y clientes tienen información de los archivos que intercambian.

Por ejemplo, si el servidor ejecuta un script PHP que genera código HTML, al cliente le enviará un archivo de texto con HTML.

La cabecera que enviará el servidor de manera automática será:

Content-type: text/html

Si un script PHP no genera código HTML se puede cambiar el tipo MIME al principio de dicho script como se verá más adelante.

8.- Subir archivos al servidor

Los formularios, además de para enviar información en forma de texto, también permiten enviar ficheros.

Al enviar archivos se debe tener en cuenta lo siguiente:

- El método ha de ser siempre **post**.
- En el formulario se debe indicar el tipo MIME mediante el atributo: **enctype="multipart/form-data"**
- El servidor tiene configurado un **tamaño máximo de archivo** de 2MB
  Se puede cambiar en el archivo php.ini (reiniciando Apache después del cambio).
  Otra manera de cambiar el tamaño máximo de archivo es añadiendo al formulario un campo oculto como el siguiente:

```
<input type="hidden" name="MAX_FILE_SIZE" value="20000">
```

- En el servidor los ficheros se almacenan en un **directorio temporal** y **cuando acaba la ejecución del script que recibe los datos los ficheros se eliminan**.

8.- Subir archivos al servidor

Para enviar archivos se utilizan los **input** tipo **file** en el formulario:

```
<form action="uploading.php" method="post" enctype="multipart/form-data">
    Nombre:
    <input type="text" name="name" id="name">
    <br>
    Selecciona tu foto de perfil:
    <input type="file" name="photo" id="photo">
    <br>
    <input type="submit" value="Enviar">
</form>
```

Para enviar más de un archivo en un formulario se puede añadir el parámetro **multiple** al **input file** o añadir varios **input file**.

9.- Recibiendo archivos desde un formulario

El **tratamiento** de archivos enviados desde un formulario se debe realizar en el **script** que se indique en el atributo **action** del formulario.

En ese script se puede usar la variable superglobal **$_FILES** para acceder a la información de los archivos recibidos.

El array **$_FILES es asociativo** y tendrá tantos elementos como archivos se hayan recibido.

A su vez, **cada elemento de $_FILES será un array asociativo con toda la información de un archivo**.

9.- Recibiendo archivos desde un formulario

A su vez, **cada elemento de $_FILES será un array asociativo** con toda la información de un archivo.

- **tmp_name**: ruta del archivo temporal en el servidor.
- **name**: nombre del archivo original (en el cliente).
- **size**: tamaño en bytes del archivo.
- **type**: tipo MIME del archivo (image/gif, text/plain...).
- **error**: código de error (UPLOAD_ERR_OK si se ha subido bien).

9.- Recibiendo archivos desde un formulario

Ejemplo, dado el siguiente formulario en el script **uploading.php**:

```
<form action="# " method="post" enctype="multipart/form-data">
    Nombre:
    <input type="text" name="name" id="name"><br>
    Selecciona la foto para subirla (formato .jpg):<br>
    <input type="file" name="photo" id="photo"><br>
    <input type="submit">
</form>
```

En el script **uploading.php** se puede acceder a la información del archivo:

```
echo $_FILES['photo']['tpm_name'];
```

9.- Recibiendo archivos desde un formulario

uploading.php

```php
if (!empty($_POST)) {
    // Se comprueba si se ha producido algún error al subir el archivo
    //  La constante UPLOAD_ERR_OK contiene el valor 0
    //  Si $_FILES['photo']['error'] no es 0 es que ha habido error
    if ($_FILES['photo']['error'] != UPLOAD_ERR_OK) {
        echo 'Error: ';
        switch ($_FILES['photo']['error']) {
            case UPLOAD_ERR_INI_SIZE:
            case UPLOAD_ERR_FORM_SIZE:  echo 'El fichero es demasiado grande.';
                                        break;
            case UPLOAD_ERR_PARTIAL:    echo 'El fichero nos e ha podido subir entero.';
                                        break;
            case UPLOAD_ERR_NO_FILE:    echo 'No se ha podido subir el fichero.';
                                        break;
            default:                    echo 'Error indeterminado.';
        }
        exit();
    }

    // Si no ha habido error se comprueba que el archivo sea del tipo requerido
    if ($_FILES['photo']['type'] != 'image/jpeg') {
        echo 'Error: No se trata de un fichero .jpg/.jpeg.';
        exit();
    }
```

9.- Recibiendo archivos desde un formulario

uploading.php

```php
    // Si no ha habido error y el archivo es del tipo requerido se comprueba si
    //  si el archivo es uno recién subido al servidor (como medida de seguridad)
    if (is_uploaded_file($_FILES['photo']['tmp_name']) === true) {
        // Hay ocasiones en las que si ya existe el archivo no se debe sobreescribir
        //  en esos casos se hará la siguiente comprobación.
        $nuevaRuta = './img/fotos-perfil/'. $_FILES['photo']['name'];
        if (is_file($nuevaRuta) === true) {
            echo 'Error: Ya existe un archivo con el mismo nombre.';
            exit();
        }

        // Se procede a mover el fichero desde el directorio temporal al directorio final
        if (!move_uploaded_file($_FILES['photo']['tmp_name'], $nuevaRuta)) {
            echo 'Error: No se puede moer el fichero a su destino';
        }
    } else {
        echo 'Error: Posible ataque. Nombre: '. $_FILES['photo']['nombre'];
    }
}
```

9.- Recibiendo archivos desde un formulario

Para comprobar que se reciben datos del formulario se comprueba si el array $_POST tiene elementos.

```php
if (!empty($_POST)) {
    // Acciones
}
```

Es importante recordar que los archivos recibidos no se encuentran en $_POST. Si se quiere comprobar si se han recibido archivos se debe usar la variable $_FILES:

```php
if (!empty($_FILES)) {
    // Acciones
}
```

Tip programming

Al guardar en el servidor archivos recibidos mediante formularios es habitual también almacenar el nombre del archivo en la base de datos.

La razón de esta práctica se verá en la unidad 5.

Se requiere de un control para que los archivos no tengan caracteres extraños o contenga información sensible.

Se puede usar la función **uniqid** para generar un nombre aleatorio, añadiendo si se quiere un prefijo.

```php
$uuid = uniqid(more_entropy: true);
```

```php
$uuid = uniqid('img', true);
```

10.- Recibiendo archivos – SEGURIDAD

Recibir datos desde el cliente puede implicar riesgos de seguridad.

Primero se realiza la comprobación de errores **$_FILES['archivo']['error']**, de esta manera solo se continúa la ejecución del script php si el archivo se ha subido de manera correcta.

A continuación, se debe comprobar que el tipo de archivo recibido sea el esperado **$_FILES['archivo']['type']**.

Por último, se debe comprobar que el archivo realmente se ha subido y no se está accediendo a un archivo ya almacenado en el servidor.

10.- Recibiendo archivos – SEGURIDAD

Hace unos años existía un ataque que consistía en engañar al script que recibía el formulario para que cogiera al archivo local /etc/passwd y así acceder a información de acceso de los usuarios.

Se crearon las funciones **is_uploaded_file** y **move_uploaoded_file** para asegurarse que se trabaja con archivos cargados desde HTTP POST.

También es habitual asegurarse que no se está sobrescribiendo un archivo ya existente mediante la función **is_file**. Aunque hay casos en los que sí es necesario sobrescribir como por ejemplo al actualizar la foto de perfil.

11.- Fallos típicos

- No tener permiso sobre la ruta indicada en el archivo **php.ini** en la directiva **upload_tmp_Dir**.
- La directiva **memory_limit** en **php.ini** tiene un valor muy pequeño o menor al de la directiva **upload_max_filesize**.
- La directiva **max_execution_time** en **php.ini** tiene un valor demasiado bajo y la ejecución del script (que incluye la subida del archivo) excede dicho tiempo.
- La directiva **post_max_size** (tamaño máximo de archivo + resto de datos que se envían en el formulario) de **php.ini** tiene debe tener un valor mayor a **upload_max_filesize**.

12.- Funciones para archivos

Además de las funciones para trabajar con archivos que se han visto PHP ofrece algunas más:

- unlink
- realpath
- dirname
- is_dir
- rename
- mkdir
- rmdir

https://www.php.net/manual/es/function.realpath.php

Práctica

Actividad 5:

Oferta de trabajo currículo y foto.

13.- Procesamiento de imágenes

El tratamiento de imágenes aplicaciones web suele darse en los siguientes supuestos:

- Crear gráficos de barras.
- Añadir marca de agua a una imagen.
- Crear una imagen a menor resolución para enviarla al cliente.
- Guardar varias versiones de una imagen recibida desde el cliente para ahorrar ancho de banda: se puede mostrar la versión de menor calidad en una galería o en un avatar al lado del nombre de usuario y se puede mostrar la versión real cuando se quiera ver en detalle.
- Evitar el robo directo de imágenes (siempre se puede capturar pantalla).
- …

13.- Procesamiento de imágenes

PHP incorpora una librería para trabajar con imágenes: **GD v2.x**

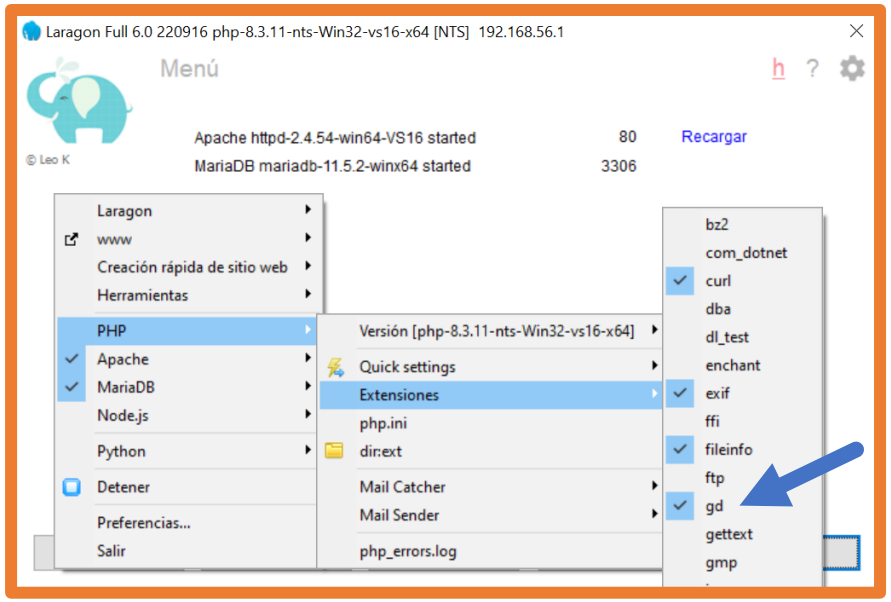
Para activarla se debe descomentar la línea siguiente (borrar el ;) del archivo php.ini:

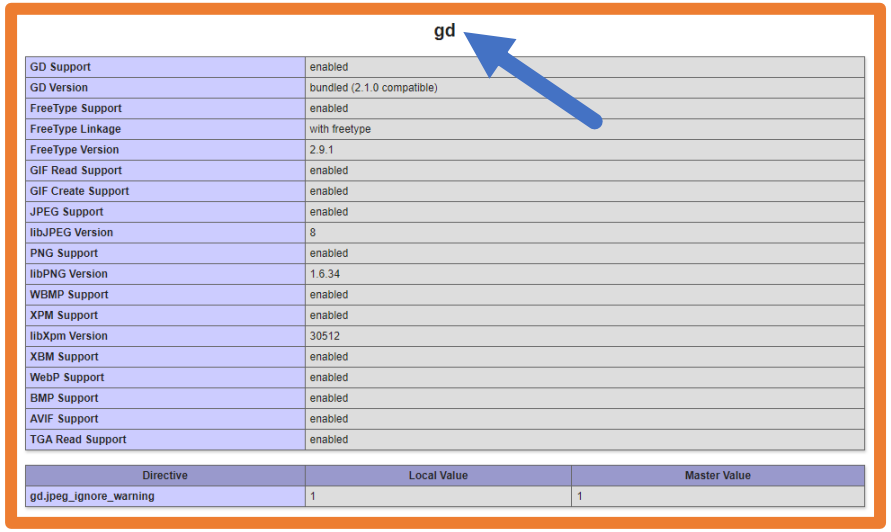
;extensión=gd

Con Laragon:

botón derecho → PHP → Extensiones → gd

Tras reiniciar Apache (Laragon lo reinicia automáticamente) mediante la función phpinfo() se puede ver la información sobre la librería





13.- Procesamiento de imágenes

Generar imágenes con GD

Hay ocasiones en los que es interesante generar imágenes desde cero, este proceso requiere de 4 pasos básicos:

- Crear lienzo sobre el que trabajar.
- Dibujar formas y/o imprimir texto en dicho lienzo.
- Generar la imagen final.
- Liberar los recursos.

13.- Procesamiento de imágenes

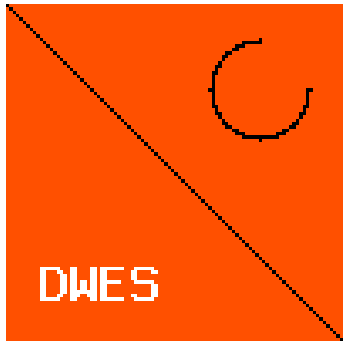
Generar imágenes con GD

```php
<?php
    // Creación del lienzo con la imagen
    $height = $width = 100;
    $image = imagecreatetruecolor($height, $width);

    // Creación de colores a utilizar sobre el lienzo
    $whiteColor = imagecolorallocate($image, 255, 255, 255);
    $blackColor = imagecolorallocate($image, 0, 0, 0);
    $orangeColor = imagecolorallocate($image, 255, 80, 0);

    // Se dibuja la imagen
    // Se rellena todo el lienzo de azul
    imagefill($image, 0, 0, $orangeColor);
    // Se crea una linea blanca desde las coordenadas (0,0) a (width,height)
    imageline($image, 0, 0, $width, $height, $blackColor);
    // Se añade una cadena al lienzo, tamaño 5 (1-5), coordenadas (5,150)
    imagestring($image, 5, 10, 75, 'DWES', $whiteColor);
    // Se añade un arco en (75,25), con diámetro (30,30) y de 0 a 270 grados
    imagearc($image, 75, 25, 30, 30, 0, 270, $blackColor);
    // Se genera la imagen
    header('content-type: image/png');
    imagepng($image);

    // Se libera la memoria
    imagedestroy($image);
```



13.- Procesamiento de imágenes

Generar imágenes con GD

La función **imagecreatetruecolor** permite crear un lienzo en blanco de las medidas que se indiquen.

Se puede crear un lienzo a partir de una imagen, de esta manera el lienzo tendrá el tamaño de la imagen que se carga:

- imagecreatefromgif
- imagecreatefromjpeg
- imagecreatefrompng
- imagecreatefromwebp

13.- Procesamiento de imágenes

Generar imágenes con GD

La función **imagecolorallocate** permite crear un color, si se quiere crear un color con nivel de transparencia existe la función **imagecolorallocatealpha**.

Una vez creado un lienzo y un color, se puede pintar en dicho lienzo usando las funciones existentes:

- imagefill
- imageline
- imagestring
- imagearc

13.- Procesamiento de imágenes

Generar imágenes con GD

La función **imagestring** permite añadir texto aunque es muy limitada.

La función **imagettftext** que permite usar fuentes (.ttf) descargadas en el servidor.

Ejemplo:

```php
imagettftext($image, 20, 0, 11, 21, $greyColor, 'arial.ttf', 'DWES');
```

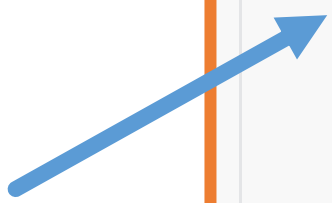
13.- Procesamiento de imágenes

Cargando imágenes con GD

En el código HTML:

```html
<img src="logo.php" alt="logo">
```

logo.php:



```php
<?php
    header('Content-Type: image/png');
    $logo = imagecreatefrompng('logo.png');
    imageAlphaBlending($logo, true);
    imageSaveAlpha($logo, true);
    imagepng($logo);
    imagedestroy($logo);
```

13.- Procesamiento de imágenes

Transformar imágenes con GD

la biblioteca GD permite transformar imágenes.

Los usos más típicos son:

- Guardar varias versiones/tamaños de la imagen.
- Poner marca de agua en las imágenes.

13.- Procesamiento de imágenes

Cambio de tamaño

```php
<?php
    // Se carga la imagen original
    $originalImage = imagecreatefrompng('dwes.png');
    // Se calculan las nuevas dimensiones: la mitad que la original
    $newWidth = intval(imagesx($originalImage)/2);
    $newHeight = intval(imagesy($originalImage)/2);
    // Se crea un lienzo con las nuevas dimensiones
    $newImage = imagecreatetruecolor($newWidth, $newHeight);
    // Se copia la imagen original en el lienzo con las medidas del lienzo
    imagecopyresized($newImage, $originalImage, 0, 0, 0, 0,
                     $newWidth, $newHeight,
                     imagesx($originalImage), imagesy($originalImage));
    // Se crea la imagen
    header('content-type: image/png');
    imagepng($newImage);
    // Se libera la memoria
    imagedestroy($originalImage);
    imagedestroy($newImage);
```

13.- Procesamiento de imágenes

Cambio de tamaño

También existe la función **imagecopyresampled** que aplica un suavizado a los bordes para obtener un resultado de mejor calidad.

Si el cambio de tamaño no respeta la relación de aspecto la imagen final se verá distorsionada.

El eje de coordenadas está en la esquina superior izquierda.

13.- Procesamiento de imágenes

Cambio de tamaño

Si simplemente se quiere escalar la imagen se usa la función **imagescale**, a la que habrá que proporcionar las nuevas medidas.

Si solo se indica el ancho el escalado será proporcional:

```php
$image = imagescale($originalImage, $newWidth, $newHeight);

// nuevas dimensiones de la imagen: 200x100
$image = imagescale($originalImage, 200, 100);

// nuevas dimensiones de la imagen: 75% de la original
$image = imagescale($originalImage, imagesx($originalImage)*0.75);

// nuevas dimensiones de la imagen: 50% de la original
$image = imagescale($originalImage, imagesx($originalImage)*0.5);
```

13.- Procesamiento de imágenes

Marca de agua

Cargar imágenes transformadas mediante la librería GD permite por ejemplo añadir una marca de agua en la imagen.

En el código **HTML**:

```
<img src="watermark.php?img=logo" alt="logo">
```

13.- Procesamiento de imágenes

Marca de agua – en el archivo **watermark.php**:

```php
<?php
    // La marca de agua puede ser una imagen existnte
    //  o una imagen creara con GD
    $watermark = imagecreatefrompng('watermark.png');
    $watermark = imagescale($watermark, 50);
    // Se indica el modo de fusión para el alpha
    imagealphablending($watermark, false);
    imagesavealpha($watermark, true);
    // Se obtienen las dimensiones de la imagen marca de agua
    $watermarkWidth = imagesx($watermark);
    $watermarkHeight = imagesy($watermark);
    // Se aplica un filtro a la marca para cambiarle el tono y la opacidad
    imagefilter($watermark, IMG_FILTER_COLORIZE, 0, 0, 0, 60);
    // Se carga la imagen sobre la que se mostrará la marca de agua y se obtienen sus dimensiones
    $image = imagecreatefromjpeg('ps5.jpg');
    $imageWidth = imagesx($image);
    $imageHeight = imagesy($image);
    // Se añade la marca de agua a la imagen
    imagecopy($image, $watermark, $watermarkWidth, $watermarkHeight, 0, 0, $imageWidth, $imageHeight);
    // Se indica la cabecerea para enviar una imagen
    header('content-type: image/png');
    // Se envía la imagen
    imagepng($image);
    // Se liberan los recuros
    imagedestroy($image);
    imagedestroy($watermark);
```

13.- Procesamiento de imágenes

Guardar imágenes

Hasta ahora se han tratado las imágenes enviándolas al cliente.

En ocasiones se puede trabajar con imágenes sin necesidad de enviar la imagen al cliente, por ejemplo, cuando se sube la foto de perfil desde un formulario.

Para guardar una imagen se utilizan las mismas funciones que para mostrarlas pero añadiendo la ruta destino:

```
imagejpeg($image, 'img/foto.jpg');
imagepng($image, 'img/foto.png');
```

Si no se va a enviar la imagen al cliente no será necesario indicar la cabecera:

```
header("Content-type: image/png");
```

Práctica

Actividad 6:

Oferta de trabajo final.

Actividad 7:

Catálogo on-line.