



UD5. Procesamiento de Formularios

Desarrollo Web en Entorno Servidor

Profesora: Silvia Vilar Pérez

curso 2024-2025

Contenidos

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

Protocolo HTTP

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

Formularios

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

Formularios - Peticiones GET

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

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

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

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

Peticiones GET - URL

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

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

Peticiones GET - URL

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

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

Peticiones GET - URL

Ejemplo:

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

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

Esto genera:

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

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

```
C%C3%A1ndido+Garc%C3%A1+Sanchez&uid=25%4012%233
```


Formularios - Peticiones POST

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

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

Formularios – POST VS GET

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

Formularios – Resumen

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

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

Formularios – HTTP GET vs POST

El método **GET** de HTTP:

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

El método **POST** de HTTP:

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

Acceso a controles de formulario desde PHP – Ejemplo

Acceso a los datos de un formulario HTML:

- Fichero uno.html

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

- Fichero dos.php

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

Acceso a controles de formulario desde PHP – Ejercicio

- Formulario HTML:

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

- PHP (¿qué obtenemos en cada caso?)

```
<?
```

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

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

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

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

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

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


Subida de archivos al servidor

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

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

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

Subida de archivos al servidor

Límite de tamaño en php.ini

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

Subida de archivos al servidor

Límite de tamaño en el propio formulario

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

Ejemplo:

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

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

Subida de archivos al servidor

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

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

Subida de archivos al servidor

Aspectos a tener en cuenta:

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

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

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

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

Subida de archivos al servidor - Ejemplo

Código del formulario

```
<html>
```

```
<body>
```

Inserción de la fotografía del usuario:

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

```
<?php
```

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

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

```
?>
```

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

```
</form>
```

```
</body>
```

```
</html>
```


Código de inserta.php

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

Procesamiento de un formulario en único fichero

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

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

Procesamiento de un formulario en único fichero

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

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

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

entonces la condición anterior se transforma en:

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

o bien

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

Validación de formularios

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

si se ha enviado el formulario

Validar datos

si hay errores

Mostrar formulario con errores

si no

Procesar formulario

fsi

si no

Mostrar formulario con valores por defecto o ya enviados

fsi

Validación formulario no vacío

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

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

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

Validación - Ejemplo

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

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

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


Validación - Ejemplo

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

Ante la entrada de texto Silvia<Vilar>

El código:

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

Obtendría el resultado

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

Validación - Ejemplo

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

Ante una entrada de usuario con espacios en blanco

El código

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

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

Validación – Ejemplo

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

Con la entrada de nombre Silvia & <Vilar>

El código

?php

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

```
echo $salida;
```

```
?>
```

Devolvería
Silvia & <Vilar>

Fichero de validación de datos

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

Ejemplo valida.php

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

```

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

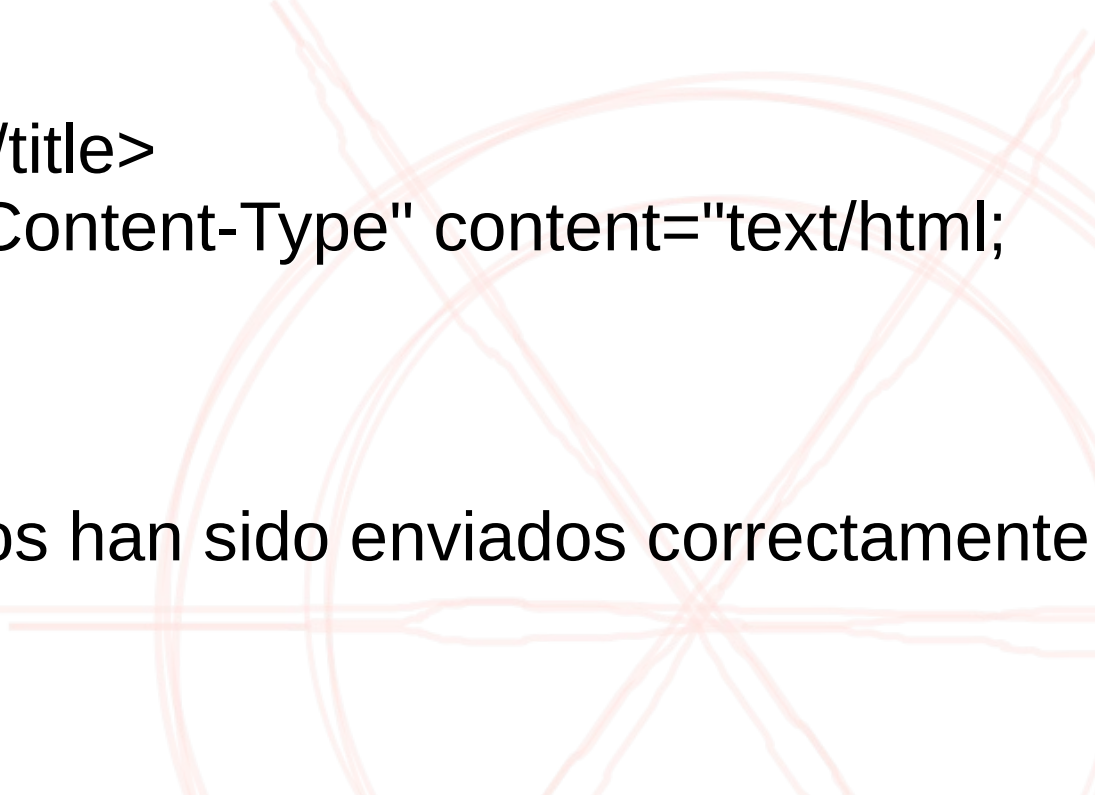
```


index.php (continuación)

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

Ejemplo validado.php

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



Funciones para comprobación de datos

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

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

Funciones is_

isset() pertenecería a este grupo de funciones

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

Funciones ctype_

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

Funciones Filter_

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

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

Funciones _exists

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

Expresiones Regulares

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

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

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

Ejemplo preg_match()

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

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

print_r($matches);
?>
```

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

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

Nota: PHP usa Perl Compatible Regular Expressions (PCRE)

Ejemplo preg_match()

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

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

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

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

Nomenclatura en ExpReg
'/' - Limitador de expresión regular

Otras Funciones con Expresiones Regulares

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

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

Expresiones Regulares

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

<https://regex101.com/>

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

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

<https://regexr.com/>

Expresiones Regulares - Ejemplos

```
<?php
```

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

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

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

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

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

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

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

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

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

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

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

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

```
?>
```

Cabeceras HTTP

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

Cabeceras HTTP

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

- Ejemplos:

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

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

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

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

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

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

Cabeceras HTTP

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

- Ejemplo:

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

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

Cabeceras HTTP – Usos frecuentes

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

Cabeceras HTTP – Usos frecuentes

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

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