



UNIVERSIDAD  
DE GRANADA



## Tecnologías Web

Grado en Ingeniería Informática

### Tema 3 – Programación en el lado del servidor El lenguaje PHP

*Este documento está protegido por la Ley de Propiedad Intelectual (Real Decreto Ley 1/1996 de 12 de abril).  
Queda expresamente prohibido su uso o distribución sin autorización del autor.*

© Javier Martínez Baena  
jbaena@ugr.es

Departamento de Ciencias de la  
Computación e Inteligencia Artificial  
<http://decsai.ugr.es>



UNIVERSIDAD  
DE GRANADA

## Tecnologías Web

Grado en Ingeniería Informática

Programación en el lado del servidor



### 1. El lenguaje PHP

1. Introducción
  2. Variables y tipos
  3. Cadenas
  4. Arrays
  5. Estructuras de control
  6. Funciones
  7. Gestión de errores
  8. Expresiones regulares
  9. Ficheros
2. PHP y aplicaciones web
  3. PHP y conexión con BBDD





## Programación en el lado del servidor

¿Porqué es necesaria?

¿Porqué hace falta la programación en el lado del servidor? (BACK-END)



Procesamiento en el servidor:

- Generación de código **HTML dinámico** en el servidor
- Acceso a recursos de cálculo: **CPU**
- Acceso a recursos de almacenamiento: **Bases de Datos**, ficheros
- Acceso a otros recursos
- Comunicación con **otros servicios**: POP3, IMAP, HTTP, ...



## Programación en el lado del servidor

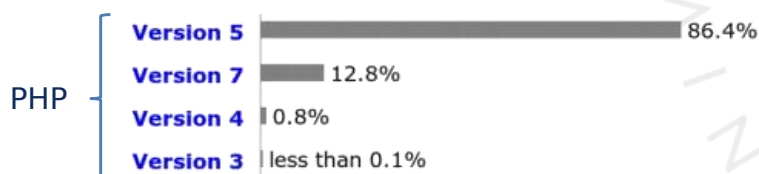
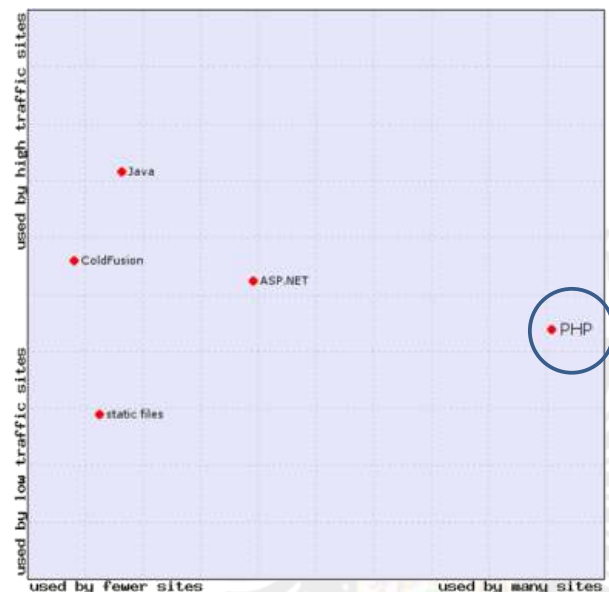
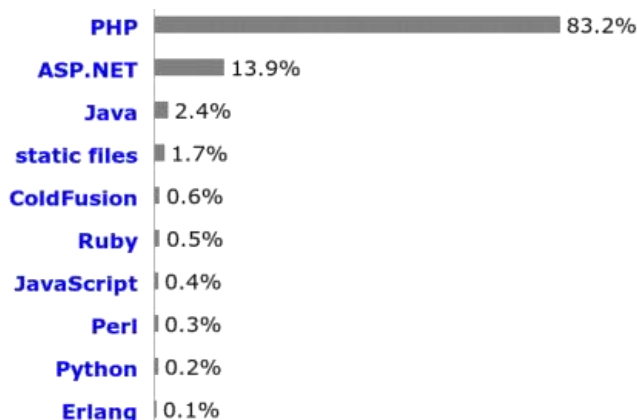
Uso de lenguajes server-side en sitios web

**W3Techs**

Web Technology Surveys

<https://w3techs.com/>

Análisis de top 10.000.000 sites  
Marzo 2018





## Programación en el lado del servidor

### Historia de PHP

PHP = "PHP: Hypertext Preprocessor"  
(inicialmente "Personal Home Page Tools")

Creado por Rasmus Lerdorf en 1994



- PHP 1.- 1995. Sale primera versión
- PHP 2.- 1997
- PHP 3.- 1998.  
Renombrado a PHP: Hypertext Preprocessor.  
Parser reescrito por Zeev Suraski y Andi Gutmans.  
Zend Engine = núcleo del intérprete PHP
- PHP 4.- 2000
- ➔ **PHP 5.- 2004.** Incluye POO
- PHP 6.- Intento de dar soporte completo Unicode. Abandonado
- PHP 7.- 2014. Mejoras de rendimiento. Menos extendido

<http://php.net/manual/es/history.php.php>

<https://en.wikipedia.org/wiki/PHP>

No hay una especificación formal

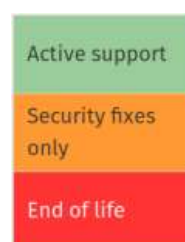
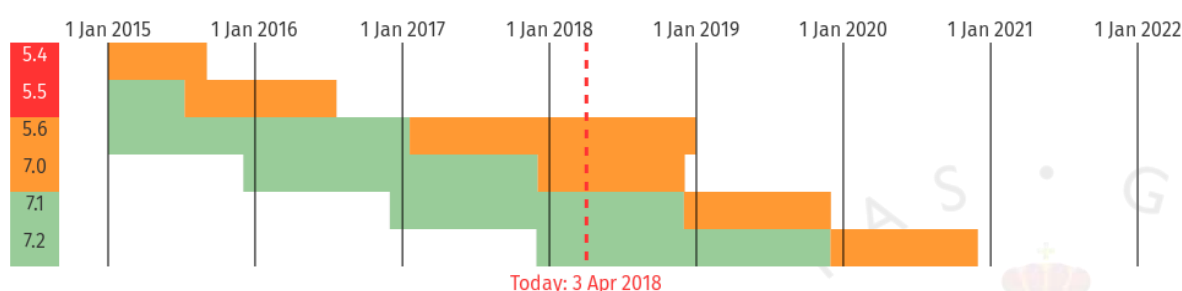
Desarrollado por The PHP Group (<http://www.php.net>)



## Programación en el lado del servidor

### Historia de PHP

#### Versiones de PHP y ciclo de vida



2 años de soporte activo (bugs + seguridad)

1 año de soporte (errores críticos de seguridad)



## El lenguaje PHP

Marcas de PHP

### Cómo escribir código PHP

- Fichero con extensión .php
- Inclusión de código entre las marcas `<?php` `?>`

hola.php

```
<?php
echo "Hola mundo";
?>
```

- El fichero puede incluir HTML y PHP conjuntamente

hola2.php

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo PHP</title>
  </head>
  <body>
    <?php echo "Hola mundo\n"; ?>
  </body>
</html>
```

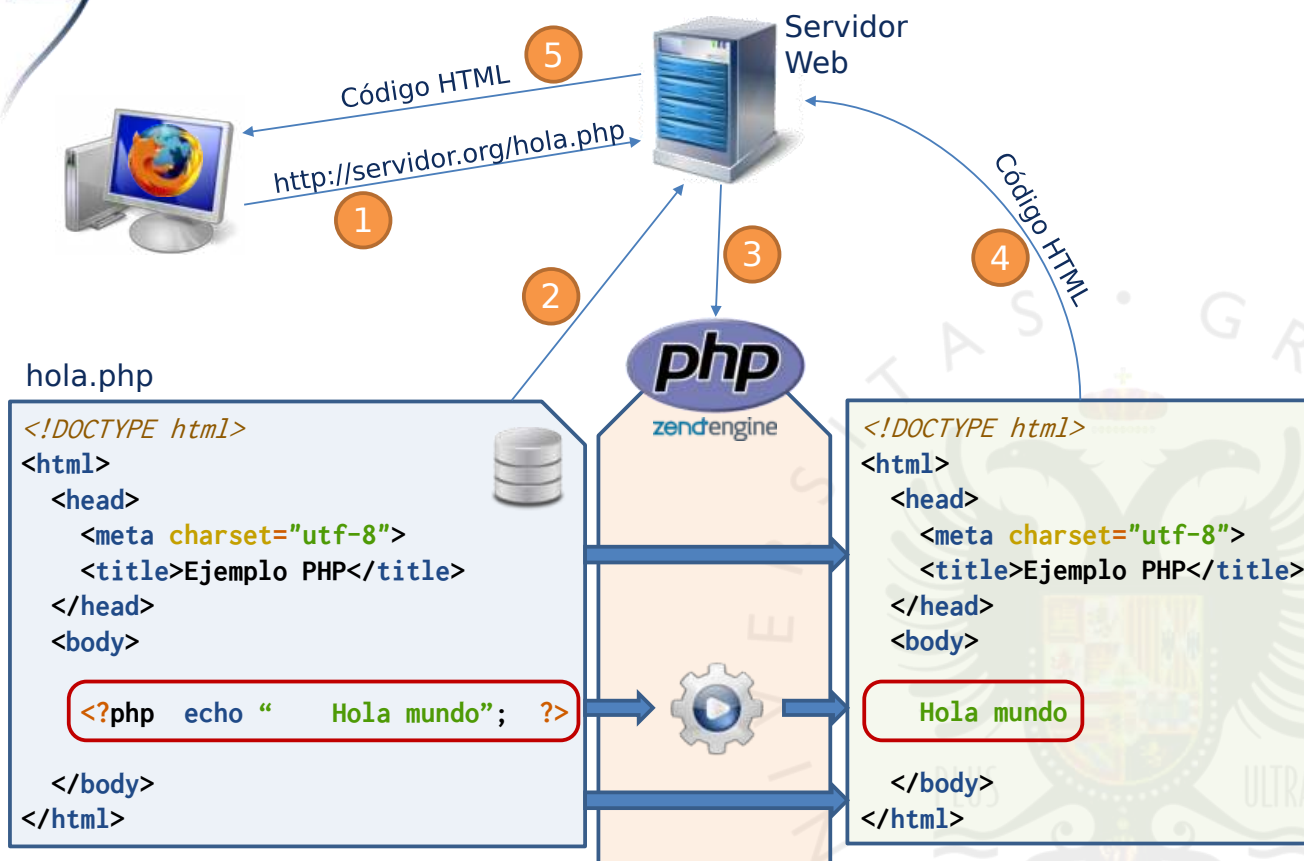
Documento devuelto

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo PHP</title>
  </head>
  <body>
    Hola mundo
  </body>
</html>
```



## El lenguaje PHP

Marcas de PHP





### ¿Cuál es más recomendable?

1

```
<body>
  <?php for ($i = 1; $i <= 10; $i++) { ?>
    <p>Párrafo número <?php echo $i; ?></p>
  } ?>
</body>
```

2

```
<body>
  <?php for ($i = 1; $i <= 10; $i++) {
    echo "<p>Párrafo número $i </p>\n";
  } ?>
</body>
```

```
<body>
  <p>Párrafo número 1</p>
  <p>Párrafo número 2</p>
  <p>Párrafo número 3</p>
  <p>Párrafo número 4</p>
  <p>Párrafo número 5</p>
  <p>Párrafo número 6</p>
  <p>Párrafo número 7</p>
  <p>Párrafo número 8</p>
  <p>Párrafo número 9</p>
  <p>Párrafo número 10</p>
</body>
```

- 1 ligeramente más eficiente que 2 (poca diferencia)  
mejor edición (coloreado de tags, tabulaciones, cierres, etc)
- 2 más legible el código PHP / menos legible el código HTML

Criterio para decidir: legibilidad, extensión, ...



### Inclusión de código de otros ficheros

Hay dos instrucciones para incluir código de otros ficheros:

- `require "fichero"` Si el fichero no existe provoca error
- `include "fichero"` Si el fichero no existe provoca solo warning

Si se incluye un mismo fichero múltiples veces el código queda duplicado:

- `require_once`, `include_once` (la segunda inclusión no tiene efecto)

```
<?php include "header.html"; ?>
<h1>El título</h1>
<?php include "footer.html"; ?>
```

header.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
```

footer.html

```
<footer>
<p>(C) Pepito Pérez</p>
</footer>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
</head>
<body>
<h1>El título</h1>
<footer>
<p>(C) Pepito Pérez</p>
</footer>
</body>
</html>
```





## El lenguaje PHP

### Inclusión de ficheros

imain.php

```
<?php
require "ihtml.php";
htmlIni("Ejemplo");
echo "Hola mundo";
htmlFin();
?>
```

ihtml.php

```
<?php

function htmlIni($titulo) {
    echo '<!DOCTYPE html> <html> <head> <meta charset="utf-8">';
    echo "<title>$titulo</title> </head> <body>";
}

function htmlFin() {
    echo '</body> </html>';
}

?>
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
    <title>Hola mundo</title>
</head>
<body>
</body>
</html>
```



## El lenguaje PHP

### Inclusión de ficheros

Inclusión de código de otros ficheros

`get_included_files()` devuelve un array con los ficheros incluidos

```
<!DOCTYPE html>
...
<?php
include_once 'f1.php';
include 'f2.html';
require_once 'f3.html';
require 'f3.html';
include_once 'f2.html';
$ar = get_included_files();
foreach ($ar as $nombre) {
    echo "$nombre<br>";
}
?>
</body>
</html>
```

```
<?php echo "<h3>Texto de prueba 1</h3>"; ?>
```

```
<h3>Texto de prueba 2</h3>
```

```
<h3>Texto de prueba 3</h3>
```

**Texto de prueba 1**  
**Texto de prueba 2**  
**Texto de prueba 3**  
**Texto de prueba 3**

```
/home/jbaena/TecnologiasWeb/src/php03.php
/home/jbaena/TecnologiasWeb/src/f1.php
/home/jbaena/TecnologiasWeb/src/f2.html
/home/jbaena/TecnologiasWeb/src/f3.html
```



## El lenguaje PHP

### Aspectos genéricos

#### Aspectos básicos

- Comentarios:
  - `//` (comentarios estilo C++)
  - `/* */` (comentarios estilo C)
  - `#` (comentarios estilo Perl)
- Usa punto y coma para separar instrucciones
- Case sensitive
  - Las variables son sensibles a mayúsculas
  - Las funciones, clases y palabras reservadas NO lo son

#### Mostrar mensajes en salida

- Instrucción `echo` (o `print`)
- Salto de línea: `"\n"` o `PHP_EOL` → (Útil para ejecución en línea de órdenes)
- `printf()`

```
<?php echo "Hola mundo", PHP_EOL;
echo "Hola mundo\n";
echo "Hola ", "mundo", PHP_EOL; // Múltiples argumentos
print "Hola mundo\n";           // Solo un argumento
echo("Hola mundo\n");           // También con paréntesis
print("Hola mundo\n"); ?>
```

<https://secure.php.net/manual/en/langref.php>



## Tecnologías Web

### Grado en Ingeniería Informática

#### Programación en el lado del servidor

#### 1. El lenguaje PHP

##### 1. Introducción



##### 2. Variables y tipos

##### 3. Cadenas

##### 4. Arrays

##### 5. Estructuras de control

##### 6. Funciones

##### 7. Gestión de errores

##### 8. Expresiones regulares

##### 9. Ficheros

#### 2. PHP y aplicaciones web

#### 3. PHP y conexión con BBDD





### Variables y tipos

- Tipificación débil
- Los nombres de las variables comienzan por \$
- Las variables son sensibles a mayúsculas

### Tipos de datos:

- Escalares:
  - Enteros
  - Reales
  - Cadenas
  - Booleanos
- Compuestos:
  - Arrays
  - Objetos
- Especiales:
  - Resource
  - NULL

```
<?php
$numero = 10;
$valor = "Cadena";
$NUMERO = 3.1415;

$v4 = TRUE;
$v5 = true;

$vec = array("primero", "segundo", "tercero");

$vec = $valor;
?>
```



### Números en PHP

- Los enteros están en el rango del tipo long de C  
-2.147.483.648 a 2.147.483.647 (para 32 bits)
- Si se intenta asignar un valor fuera del rango de los enteros se convierte a floating-point automáticamente.
- Función `is_int()` comprueba si el argumento es entero
- Función `is_float()` comprueba si el argumento es real
- Función `is_real()` comprueba si el argumento es real (alias de `is_float`)

```
<?php
$a = 9223372036854775807; // Máximo entero con 64 bits
$b = 9223372036854775808; // Overflow ?
if (is_int($a)) {
    echo "a es entero", PHP_EOL;
}
if (is_int($b)) {
    echo "b es entero", PHP_EOL;
}
echo $a, PHP_EOL;
echo $b, PHP_EOL;
?>
```

```
jbaena@void: ~/public_html/test
jbaena@void:~/public_html/test$ php numeros.php
a es entero
9223372036854775807
9.2233720368548E+18
jbaena@void:~/public_html/test$
```





## Tipos de datos

### Datos numéricos

#### Operadores aritméticos

- ++, -- (pre/post incremento/decremento)
- \*, /, % (multiplicación, división, módulo)
- +, - (suma, resta)

#### Operadores a nivel de bit (también con otros tipos)

- ~ Negación
- &, |, ^ Y, O, XOR
- <<, >> Desplazamiento de bits (inserta ceros)

#### Asignación (=)

- Devuelve el valor que ha asignado (asignaciones en cadena)
- Copy-on-write: las asignaciones entre variables son referencias. Cuando se hacen modificaciones se duplican.
- PHP tiene recolector de basura

#### Asignaciones con operación

- +=, -=, \*=, /=, % =
- ^=, &=, |=



## Tipos de datos

### Datos booleanos y NULL

#### Datos booleanos

- Literales: true, false (no es sensible a mayúsculas/minúsculas)
- Evalúan a falso estos valores:
  - False
  - 0
  - 0.0
  - ""
  - "0"
  - Array con cero elementos
  - NULL
- Función is\_bool() comprueba si el argumento es booleano

#### El tipo NULL

- Solo existe un valor para este tipo: NULL
- Representa a una variable que no tiene valor
- Función is\_null() comprueba si el argumento es NULL



## Tipos de datos

### Operadores relacionales y lógicos

#### Operadores relacionales (todos de igual prioridad)

- `==, !=, <>` Igualdad/desigualdad de los valores
- `<, >, <=, >=` Orden
- `===, !==` Igualdad/desigualdad de los valores y del tipo

```
2==2 // true
```

```
2===2 // true
```

```
2==2.0 // true (hay conversión de tipos)
```

```
2===2.0 // false
```

```
true==false // false
```

```
false==0 // true (hay conversión de tipos)
```

```
false===0 // false
```

#### Operadores lógicos (ordenados por precedencia)

- `!`
- `and, &&`
- `xor`
- `or, ||`

*PHP implementa evaluación en corto (short-circuit)*



## Tipos de datos

### Información de variables

#### Información sobre variables

<code>var_dump(expr)</code>	Muestra el tipo y valor de una expresión
<code>isset(\$v)</code>	Comprueba la existencia de una variable
<code>unset(\$v)</code>	Elimina una variable
<code>gettype(\$v)</code>	Devuelve el tipo de una variable (como cadena)

```
$v1 = 23;
```

```
var_dump($v1); // int(23)
```

```
$v2 = 17.56;
```

```
var_dump($v2); // float(17.56)
```

```
$v3 = true;
```

```
var_dump($v3); // bool(true)
```

```
$v4 = NULL;
```

```
var_dump($v4); // NULL
```

```
echo gettype($v1), PHP_EOL; // "integer"
```

```
echo gettype($v2), PHP_EOL; // "double"
```

```
echo gettype($v3), PHP_EOL; // "boolean"
```

```
var_dump(isset($v)); // bool(false)
```

```
$v = 23.12;
```

```
var_dump(isset($v)); // bool(true)
```

```
unset($v);
```

```
var_dump(isset($v)); // bool(false)
```



## Tipos de Datos

### Constantes

#### Constantes

- Definición de constantes
  - `define("NOMBRE", "VALOR");` Definición en tiempo de ejecución
  - `const NOMBRE = VALOR;` Definición en tiempo de compilación
- Recomendación: usar mayúsculas

```
define("CTE1", "valor1");
define("CTE2", true);
define("CTE3", 3);
define(CTE4, 7);

const CTE5 = 6;
const CTE6 = "Hola";

echo CTE4, PHP_EOL;
echo CTE5, PHP_EOL;
```

Constantes predefinidas (*Magic constants*): tienen información sobre el código PHP (usadas para depurar):

```
__LINE__
__FILE__
__DIR__
__FUNCTION__
__CLASS__
__METHOD__
__NAMESPACE__
```



## Tipos de Datos

### Conversiones explícitas e implícitas

#### Conversiones implícitas

- Integer op float : se hace casting a float
- Número op string : el string se convierte a número

```
$x = 4 + 4.5;           // resultado float 8.5
$x = 3 / 2;            // resultado float 1.5
$x = 3 + "6";          // resultado integer 9
$x = 3 + "4 manzanas"; // resultado integer 7;
```

#### Conversiones explícitas

```
(int)    (integer)
(bool)   (boolean)
(float)  (double)  (real)
(string)
```

```
$x = (int) 4.5;          // asigna 4
$x = (int)(3 / 2);      // asigna 1
```

#### 1. El lenguaje PHP

##### 1. Introducción

##### 2. Variables y tipos



##### 3. Cadenas

##### 4. Arrays

##### 5. Estructuras de control

##### 6. Funciones

##### 7. Gestión de errores

##### 8. Expresiones regulares

##### 9. Ficheros

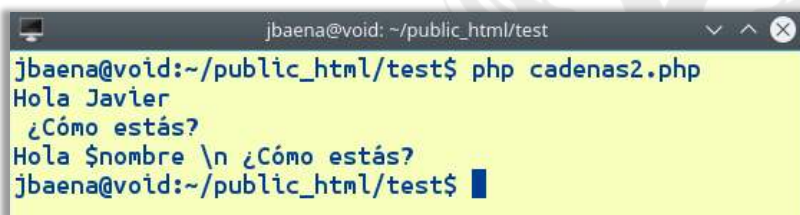
#### 2. PHP y aplicaciones web

#### 3. PHP y conexión con BBDD

#### Cadenas de caracteres

- Los literales pueden ir entre comillas:
  - " " Dobles: Se sustituye el valor de las variables incluidas y permite el uso de códigos de escape (\n, \t, \\\, ...)
  - ' ' Simples: No se sustituye el valor de las variables incluidas
- Función `is_string()` comprueba si el argumento es una cadena

```
<?php
$nombre = "Javier";
$saluda1 = "Hola $nombre \n ¿Cómo estás?";
$saluda2 = 'Hola $nombre \n ¿Cómo estás?';
echo $saluda1, PHP_EOL;
echo $saluda2, PHP_EOL;
?>
```



```
jbaena@void: ~/public_html/test
jbaena@void:~/public_html/test$ php cadenas2.php
Hola Javier
¿Cómo estás?
Hola $nombre \n ¿Cómo estás?
jbaena@void:~/public_html/test$
```

```
$palabra = "Coche";
echo "El plural de $palabra es {$palabra}s", PHP_EOL;
```



## Cadenas de caracteres

### Comparación

#### Comparación de cadenas

- Operadores relacionales
  - === y !== no hacen conversión de tipos
  - El resto sí hacen conversión de tipos

*// Comparación de cadenas con números*

```
echo ("12"==12 ? "true" : "false"), PHP_EOL; // T Comparación numérica
echo ("12"=="12" ? "true" : "false"), PHP_EOL; // T Comparación numérica
echo ("12"=="12a" ? "true" : "false"), PHP_EOL; // F Comparación texto
echo (12=="12a" ? "true" : "false"), PHP_EOL; // T Comparación numérica
echo ("0"=="0.0" ? "true" : "false"), PHP_EOL; // T Comparación numérica
echo ("0"=== "0.0" ? "true" : "false"), PHP_EOL; // F Comparación texto
```

#### Similaridad entre cadenas

- soundex, metaphone: se usan para buscar similitudes según la fonética de las cadenas
- similar\_text: cuenta el número de caracteres en común entre cadenas
- levenshtein: cuenta el número de caracteres que habría que añadir, insertar o borrar para que coincidan dos cadenas



## Cadenas de caracteres

### Comparación

#### Comparación de cadenas

- Comparación de cadenas: strcmp, strcasecmp
- Comparación natural de cadenas: strnatcmp, strnatcasecmp

*// Comparación de cadenas de texto*

```
$cad1 = "Javier";
$cad2 = "jaViER";
echo (strcmp($cad1,$cad2)==0 ? "true" : "false"), PHP_EOL; // false
echo (strcasecmp($cad1,$cad2)==0 ? "true" : "false"), PHP_EOL; // true
```

*// Comparación "natural"*

```
$cad1 = "fichero1.txt";
$cad2 = "fichero3.txt";
$cad3 = "fichero15.txt";

echo strcmp($cad1,$cad2), PHP_EOL; // <0
echo strcmp($cad2,$cad3), PHP_EOL; // >0

echo strnatcmp($cad1,$cad2), PHP_EOL; // <0
echo strnatcmp($cad2,$cad3), PHP_EOL; // <0
```





## Cadenas de caracteres

### Caracteres, concatenación

```
// Acceso a caracteres individuales
$nombre = "javier";
echo $nombre[0], $nombre[1], $nombre[2], PHP_EOL;    // jav

// Y modificación
$nombre[0] = "J";
echo $nombre, PHP_EOL;    // Javier

// Longitud de la cadena
echo strlen($nombre), PHP_EOL;    // 6

// Concatenación
$apellido1 = "Martínez";
$apellido2 = "Baena";
$completo = $nombre . ' ' . $apellido1;    // Operador .
$completo .= ' ' . $apellido2;    // Operador .=
echo $completo, PHP_EOL;    // Javier Martínez Baena

// Eliminar espacios (blanco, tab, enter, etc)
$cadena = " \t Javier Martínez ";
echo "cadena >", $cadena, "<", PHP_EOL;
echo "ltrim >", ltrim($cadena), "<", PHP_EOL;    // "Javier Martínez "
echo "rtrim >", rtrim($cadena), "<", PHP_EOL;    // "   Javier Martínez"
echo "trim >", trim($cadena), "<", PHP_EOL;    // "Javier Martínez"
```



## Cadenas de caracteres

### Extracción, conteo

```
// Conversión mayúsculas/minúsculas
$cadena = "jAVier MarTÍNEZ bAeNa";
echo "cadena : ", $cadena, PHP_EOL;
echo "strtolower: ", strtolower($cadena), PHP_EOL;    // javier martínez baena
echo "strtoupper: ", strtoupper($cadena), PHP_EOL;    // JAVIER MARTÍNEZ BAENA
echo "ucfirst: ", ucfirst($cadena), PHP_EOL;    // JAVier MarTÍNEZ bAeNa
echo "ucwords: ", ucwords($cadena), PHP_EOL;    // JAVier MarTÍNEZ bAeNa
echo "lcfirst: ", lcfirst($cadena), PHP_EOL;    // jAVier MarTÍNEZ bAeNa

// Extraer subcadenas
$cadena = "Ford Mustang";
echo substr($cadena,0,4), PHP_EOL;    // Ford (inicio - longitud)
echo substr($cadena,5), PHP_EOL;    // Mustang (inicio - hasta final)
// Si inicio es negativo: es relativo al final de la cadena
echo substr($cadena,-7), PHP_EOL;    // Mustang
// Si longitud es negativo: quitar esos caracteres desde el final
echo substr($cadena,5,-2), PHP_EOL;    // Musta

// Contar ocurrencias de una subcadena en otra
$cadena = "Juan Col y Col de la Col" ;
echo substr_count($cadena,"Col"), PHP_EOL;    // 3
echo substr_count($cadena,"Col",8), PHP_EOL;    // 2
echo substr_count($cadena,"Col",8,9), PHP_EOL;    // 1
```



## Cadenas de caracteres

### Reemplazo

*// Reemplazar subcadenas*

```
$cadena = "Juan Garcia Sanchez";
echo substr_replace($cadena,"Cantalejo",5,6), PHP_EOL; // Juan Cantalejo Sanchez
echo substr_replace($cadena,"Cantalejo",5), PHP_EOL; // Juan Cantalejo
echo substr_replace($cadena,"Cantalejo",-7), PHP_EOL; // Juan Garcia Cantalejo
```

*// Cuidado con caracteres UTF-8*

```
$cadena = "Juan García Sánchez";
echo substr_replace($cadena,"López",5,6), PHP_EOL; // Juan López Sánchez
echo substr_replace($cadena,"López",5,7), PHP_EOL; // Juan López Sánchez
```

*// Para insertar/borrar una subcadena*

```
$cadena = "Juan García Sánchez";
echo substr_replace($cadena,"Carlos ",5,0), PHP_EOL; // Juan Carlos García Sánchez
echo substr_replace($cadena,"",5,8), PHP_EOL; // Juan Sánchez
```

*// Reemplazo de múltiples ocurrencias*

```
$cad = "Juan Col y Col de Tomate";
echo str_replace("Col","Apio",$cad,$num), PHP_EOL; // Juan Apio y Apio de Tomate
echo "Sustituciones: $num", PHP_EOL; // 2
```

*// No sensible a mayúsculas*

```
echo str_ireplace("col","Apio",$cad,$num), PHP_EOL; // Juan Apio y Apio de Tomate
```



## Cadenas de caracteres

### Justificación, búsqueda

*// Justificar cadenas*

```
$cadena = "Ford Mustang";
echo ">",str_pad($cadena,20),"<",PHP_EOL; // >Ford Mustang <
echo ">",str_pad($cadena,20," ",STR_PAD_LEFT),"<",PHP_EOL; // > Ford Mustang<
echo ">",str_pad($cadena,20," ",STR_PAD_BOTH),"<",PHP_EOL; // > Ford Mustang <
echo ">",str_pad($cadena,20,".",STR_PAD_RIGHT),"<",PHP_EOL; // >Ford Mustang.....<
```

*// Buscar primera ocurrencia de subcadena*

```
$cadena = "Juan Col y Col de Tomate";
echo strpos($cadena,"Col"), PHP_EOL; // 5
echo strpos($cadena,"Col",6), PHP_EOL; // 11
echo strpos($cadena,"col"), PHP_EOL; // 5 (no sensible a M/m)
```

*// Buscar última ocurrencia de subcadena*

```
echo strrpos($cadena,"Col"), PHP_EOL; // 11
```

PHP trabaja constantemente con cadenas ... hay muchas más funciones:

<http://php.net/ref.strings>



## Cadenas de caracteres

Here documents

### Heredoc

Simplifica la escritura de cadenas de varias líneas de texto (que, por ejemplo, puede ser código HTML)

```
<<<MARCADOR↵
```

```
...
```

```
MARCADOR;↵
```

```
<?php
$palabra = <<<FinTexto
Esto es una cadena
    que ocupa varias líneas
        de texto llamada "here-doc"
Observe que se conservan caracteres
especiales,    espacios,    saltos de línea, etc.
```

**FinTexto;**

```
echo $palabra, PHP_EOL;
?>
```

```
jbaena@void: ~/public_html/test
jbaena@void:~/public_html/test$ php heredoc1.php
Esto es una cadena
    que ocupa varias líneas
        de texto llamada "here-doc"
Observe que se conservan caracteres
especiales,    espacios,    saltos de línea, etc.
jbaena@void:~/public_html/test$
```



## Cadenas de caracteres

Here documents

### Heredoc

También ayuda a mejorar la legibilidad del documento cuando hay mezcla de PHP y HTML

```
<p>El usuario cuyo login es <?php echo $login; ?> (llamado <?php echo
$nombre; ?> <?php echo $apellidos; ?> ) nació en el año <?php echo
$fnac; ?> y su cuota de socio es <?php echo $cuota; ?>
</p>
```

```
<?php
echo "<p>El usuario cuyo login es $login; (llamado $nombre $apellidos)
nació en el año $fnac y su cuota de socio es $cuota</p>";
?>
```

```
<?php
echo <<<HTML
<p>El usuario cuyo login es $login (llamado $nombre $apellidos) nació
en el año $fnac y su cuota de socio es $cuota</p>
HTML;
?>
```



## Heredoc y Nowdoc

Heredoc es a los strings con comillas dobles lo que Nowdoc es a los string con comillas simples

```
<?php
```

```
$x=7;
```

```
$palabra1 = <<<FinTexto
```

```
La variable x vale $x
```

```
FinTexto;
```

```
echo $palabra1, PHP_EOL;
```

```
$palabra2 = <<<'FinTexto'
```

```
La variable x vale $x
```

```
FinTexto;
```

```
echo $palabra2, PHP_EOL;
```

```
?>
```

```
jbaena@void: ~/public_html/test
jbaena@void:~/public_html/test$ php heredoc2.php
La variable x vale 7
La variable x vale $x
jbaena@void:~/public_html/test$
```



UNIVERSIDAD  
DE GRANADA

# Tecnologías Web

Grado en Ingeniería Informática

Programación en el lado del servidor

## 1. El lenguaje PHP

1. Introducción

2. Variables y tipos

3. Cadenas



4. Arrays

5. Estructuras de control

6. Funciones

7. Gestión de errores

8. Expresiones regulares

9. Ficheros

2. PHP y aplicaciones web

3. PHP y conexión con BBDD



## Tipos de arrays

Colección de datos de cualquier tipo

- Indexados. La clave es un entero (comenzando en 0)
- Asociativos. La clave es un string

Internamente se tratan todos como asociativos

El orden de almacenamiento es el orden de inserción

*// Indexado*

```
$notas = array(5.9, 8.3, 1.7);
```

```
$notas[] = 6.1; // Añadir elemento al final
```

```
$notas2 = [5.9, 8.3, 1.7]; // Sintaxis alternativa
```

*// Indexado*

```
$asig[0] = "MC";
```

```
$asig[5] = "FP";
```

```
$asig[2] = "RSC";
```

```
$asig[] = "DBA"; // Añade al final (elemento "6")
```

*// Asociativo*

```
$profes = array("TW" => "Javier", "MP" => "Antonio", "ED" => "Rosa");
```

*// Crear arrays con rangos de valores*

```
$v1 = range(2,7); // 2, 3, 4, 5, 6, 7
```

```
$v2 = range("a","f"); // a, b, c, d, e, f
```

```
$v3 = range("z","x"); // z, y, w, x
```

*// Imprimir un array de forma sencilla*

```
print_r($v1);
```

*// Tamaño de un array*

```
echo sizeof($v1), PHP_EOL;
```

```
echo count($v2), PHP_EOL;
```

*// Crear un array con valores inicializados*

```
$a = array(2,4,3);
```

```
$b = array_pad($a, 6, 1); // 2, 4, 3, 1, 1, 1
```

```
$c = array_pad($a, 2, 1); // 2, 4, 3 ($a tiene más de 2 elementos)
```

```
$d = array_pad($a, -6, 1); // 1, 1, 1, 2, 4, 3
```

*// Matriz 2D*

```
$m = array(array(2,5,4,3),
```

```
            array(5,2,8,7),
```

```
            array(6,1,2,0));
```

```
echo "{$m[2][1]}", PHP_EOL; // Necesarias las llaves
```



## Arrays

## Variables y arrays

```
// Copiar valores de un array en variables
$v = array("Javier", "TW", 3);
list($prof,$asig,$curso) = $v;
list($prof,$asig) = $v;           // Solo se asignan los primeros
list($prof,$asig,$curso,$otro) = $v; // $otro=NULL
list($prof,, $curso) = $v;       // Saltando algunos items del array

// Creando automáticamente variables a partir del array
$asoc = ["asig" => "TW", "profe" => "Javier", "curso" => 3];
extract($asoc);
echo "$asig, $profe, $curso", PHP_EOL;

// Idem pero añadiendo un prefijo a nombres de variables
extract($asoc, EXTR_PREFIX_ALL, "mivar");
echo "$mivar_asig, $mivar_profe, $mivar_curso", PHP_EOL;
// EXTR_SKIP Si existía la variable no modificarla
// EXTR_IF_EXISTS Reemplazar variable si existía
// ...

// Crear array asociativo a partir de variables
$ciudad = "Granada";
$pais = "España";
$continente = "Europa";
$geo = compact("ciudad", "pais", "continente");
```

## Arrays

## Claves y valores

```
// Extraer las claves de un array asociativo
$asoc = ["asig" => "TW", "profe" => "Javier", "curso" => 3];
$cla = array_keys($asoc); // "asig", "profe", "curso"

$f = [1,2,3,4,5,6,7,8,9,10];
$cla = array_keys($f); // 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

// Extraer los valores de un array asociativo
$val = array_values($asoc); // "TW", "Javier", 3

// Comprobar si una clave existe en un vector
if (array_key_exists("asig",$asoc))
    echo "Existe la asignatura";
else
    echo "No existe la asignatura";

// Intercambiar claves por valores
$asoc = ["asig" => "TW", "profe" => "Javier", "curso" => 3];
$flip = array_flip($asoc); print_r($flip);
// [ "TW"=>"asig", "Javier"=>"profe", "3"=>"curso" ]
```



## Arrays

### Extracción, cadenas

```
// Extraer un subconjunto de elementos consecutivos de un array
$v1 = [2,3,4,5,6,7];
$sub = array_slice($v1, 2, 3); // 4, 5, 6

// Partir un array en arrays de menor tamaño
$f = range(1,10);
$trozos = array_chunk($f, 3); // [ [1,2,3] , [4,5,6] , [7,8,9] , [10] ]

// División de cadenas
$cadena = "Javier Martínez Baena";
$palabras = explode(" ", $cadena); // ["Javier", "Martínez", "Baena"]
$palabras = explode(" ", $cadena, 2); // ["Javier", "Martínez Baena"]

// Unión de cadenas (join es un alias de implode)
echo implode(" ", ["Sal", "Tomate", "Col"]); // "Sal, Tomate, Col"
```



## Arrays

### Borrar, insertar, buscar

```
// Eliminar elementos de un vector
$f = [1,2,3,4,5,6,7,8,9,10];
$borrados = array_splice($f, 2, 4);
print_r($borrados); // 3, 4, 5, 6
print_r($f); // 1, 2, 7, 8, 9, 10

// Sustituir elementos de un vector
$f = [1,2,3,4,5,6,7,8,9,10];
$borrados = array_splice($f, 2, 4, [20,21]);
print_r($borrados); // 3, 4, 5, 6 (los que quita)
print_r($f); // 1, 2, 20, 21, 7, 8, 9, 10

// Insertar elementos en un vector
$f = range(1,10);
array_splice($f, 2, 0, [20,21]);
print_r($f); // 1, 2, 20, 21, 3, 4, 5, 6, 7, 8, 9, 10

// Buscar un valor en un array
$v = [3, 6, 2, 8, 9];
if (in_array(8,$v)) // Devuelve true/false
    echo "Lo he encontrado", PHP_EOL;
else
    echo "No lo he encontrado", PHP_EOL;
$c = array_search(8,$v); // Devuelve la clave. Si no está devuelve false
echo "Resultado: $c", PHP_EOL; // $c=3
```



## Arrays Ordenación

```
// Ordenación de un vector (útil con vectores indexados)
$v = [3, 6, 2, 8, 1, 5];
sort($v); // 1, 2, 3, 5, 6, 8
rsort($v); // 8, 6, 5, 3, 2, 1
// Ambas funciones reasignan los índices

// Ordenación por valor (mantiene los pares clave/valor) (útil con v. asociativos)
$v = [3, 6, 2, 8, 1, 5];
asort($v); // "4"=>1, "2"=>2, "0"=>3, "5"=>5, "1"=>6, "3"=>8
arsort($v); // Idem pero descendente

// Ordenación por clave (mantiene los pares clave/valor) (útil con v. asociativos)
$v = ["jbaena"=>342, "agarrido"=>287, "jfv"=>186, "rosa"=>478];
ksort($v); // "agarrido"=>287, "jbaena"=>342, "jfv"=>186, "rosa"=>478
krsort($v); // Idem pero descendente

// Ordenación de cadenas numéricas
$v = ["fichero1.txt", "fichero12.txt", "fichero7.txt"];
sort($v); // fichero1.txt, fichero12.txt, fichero7.txt
natsort($v); // fichero1.txt, fichero7.txt, fichero12.txt

// Ordenación de cadenas numéricas no sensible a mayúsculas
$v = ["fichero1.txt", "ficHero12.txt", "Fichero7.txt"];
natsort($v); // Fichero7.txt, ficHero12.txt, fichero1.txt
natcasesort($v); // fichero1.txt, Fichero7.txt, ficHero7.txt
```



## Arrays Ordenación

```
// Ordenación de múltiples arrays emparejados
$nombre = ["Javier", "Antonio", "Joaquín", "Rosa"];
$despacho = [24, 18, 12, 18];
$categ = ["PTU", "PTU", "CU", "PTU"];
array_multisort($categ, SORT_ASC, $despacho, SORT_DESC, $nombre, SORT_ASC);
for ($i=0; $i<count($nombre); $i++)
    echo "$nombre[$i] - $despacho[$i] - $categ[$i]", PHP_EOL;
// Joaquín - 12 - CU
// Javier - 24 - PTU
// Antonio - 18 - PTU
// Rosa - 18 - PTU
```



## Variables globales

### Variables globales y superglobales

#### Variables globales

Cualquier variable definida fuera de una función

```
$vari = 34;
echo $vari, "<br>";           // 34
```

#### Variables superglobales: variables globales predefinidas

Variables globales predefinidas

**\$GLOBALS** Incluye todas las variables globales del usuario

```
$vari = 34;
echo $vari, "<br>";           // 34
echo $GLOBALS["vari"], "<br>"; // 34
```



## Variables globales

### Variables globales y superglobales

#### Variables superglobales: variables globales predefinidas

**\$\_SERVER** Información creada por el servidor (rutas, cabeceras, etc)

```
$_SERVER["SCRIPT_NAME"]=> "/tw/php/superglobals.php"
$_SERVER["REQUEST_METHOD"]=> "GET"
$_SERVER["QUERY_STRING"]=> ""
$_SERVER["SCRIPT_FILENAME"]=> "/var/www/html/tw/php/superglobals.php"

$_SERVER["HTTP_HOST"]=> "localhost"
$_SERVER["HTTP_USER_AGENT"]=> "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0)
    Gecko/20100101 Firefox/52.0"
$_SERVER["SERVER_SOFTWARE"]=> "Apache/2.4.18 (Ubuntu)"
$_SERVER["SERVER_NAME"]=> "localhost"
$_SERVER["SERVER_ADDR"]=> "127.0.0.1"
$_SERVER["SERVER_PORT"]=> "80"
$_SERVER["REMOTE_ADDR"]=> "127.0.0.1"
$_SERVER["DOCUMENT_ROOT"]=> "/var/www/html"
$_SERVER["SERVER_PROTOCOL"]=> "HTTP/1.1"
$_SERVER["REQUEST_URI"]=> "/tw/php/superglobals.php"
$_SERVER["PHP_SELF"]=> "/tw/php/superglobals.php"
```

*No está garantizado que cualquier servidor disponga de las mismas variables*



## Variables globales

### Variables globales y superglobales

#### Variables superglobales: variables globales predefinidas

##### Variables globales predefinidas

<code>\$_GET</code>	Parámetros pasados al script con método GET
<code>\$_POST</code>	Parámetros pasados al script con método POST
<code>\$_REQUEST</code>	Incluye las variables de <code>\$_GET</code> , <code>\$_POST</code> y <code>\$_COOKIE</code>

`http://localhost/tw/php/superglobals.php?nombre=Pepito&ape=P%C3%A9rez`

```
echo $_GET['nombre'];    // Pepito
echo $_GET['ape'];       // Pérez

echo $_REQUEST['nombre']; // Pepito
echo $_REQUEST['ape'];    // Pérez
```

<code>\$_FILES</code>	Ficheros subidos por HTTP POST (desde formularios)
<code>\$_COOKIE</code>	Variables pasadas al script mediante HTTP cookies
<code>\$_SESSION</code>	Variables de sesión disponibles
<code>\$_ENV</code>	Variables de entorno del intérprete de PHP



## Variables globales

### Variables globales y superglobales

#### php.ini

##### Configuración de variables superglobales en el servidor

```
; This directive determines which super global arrays are registered when PHP
; starts up. G,P,C,E & S are abbreviations for the following respective super
; globals: GET, POST, COOKIE, ENV and SERVER. There is a performance penalty
; paid for the registration of these arrays and because ENV is not as commonly
; used as the others, ENV is not recommended on productions servers. You
; can still get access to the environment variables through getenv() should you
; need to.
; Default Value: "EGPCS" / ; Development Value: "GPCS" / ; Production Value: "GPCS";
; http://php.net/variables-order
variables_order = "GPCS"

; This directive determines which super global data (G,P & C) should be
; registered into the super global array REQUEST. If so, it also determines
; the order in which that data is registered. The values for this directive
; are specified in the same manner as the variables_order directive,
; EXCEPT one. Leaving this value empty will cause PHP to use the value set
; in the variables_order directive. It does not mean it will leave the super
; globals array REQUEST empty.
; Default Value: None / ; Development Value: "GP" / ; Production Value: "GP"
; http://php.net/request-order
request_order = "GP"
```



#### 1. El lenguaje PHP

1. Introducción
2. Variables y tipos
3. Cadenas
4. Arrays
- » 5. Estructuras de control
6. Funciones
7. Gestión de errores
8. Expresiones regulares
9. Ficheros

#### 2. PHP y aplicaciones web

#### 3. PHP y conexión con BBDD

#### Estructuras de control condicionales:

- If / else / elseif
- switch

```
<?php
$sistema = "Linux";
if ($sistema=="Linux") {
    echo "estás usando linux", PHP_EOL;
} elseif ($sistema=="Windows") {
    echo "estás usando windows", PHP_EOL;
} else {
    echo "estás usando otro sistema", PHP_EOL;
}
?>
```

```
<?php
$sistema = "Linux";
switch ($sistema) {
    case "Linux":
        echo "estás usando linux", PHP_EOL;
        break;
    case "Windows":
        echo "estás usando windows", PHP_EOL;
        break;
    default:
        echo "estás usando otro sistema", PHP_EOL;
}
?>
```



## Estructuras de control

### Iterativas

#### Tipos de bucles

- for
- while
- do-while
- foreach

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

```
$i = 1;
do {
    echo "$i ";
    $i ++;
} while ($i<=10);
```

```
for ($i=1; $i<=10; $i++)
    echo "$i ";
```

```
$v1 = array("primero", "segundo", "tercero");
foreach ($v1 as $item)
    echo "$item ";
```

```
$v2 = array("primero" => "Sopa",
            "segundo" => "Solomillo",
            "tercero" => "Flan");
foreach ($v2 as $clave => $valor)
    echo "$clave : $valor ";
```



## Estructuras de control

### Iterativas

Foreach trabaja con una copia del vector para recorrerlo

```
$v = array("ciudad" => "Granada",
           "pais" => "España",
           "continente" => "Europa");
```

```
foreach ($v as $clave => $valor) {
    echo "$clave => $valor", PHP_EOL;
    // Borra todos los elementos
    array_splice($v, 0);
}
print_r($v);
```

```
foreach ($v as $clave => $valor) {
    echo "$clave => $valor", PHP_EOL;
    // Añade elemento
    $v["{$clave}_x"] = "procesado";
}
print_r($v);
```

```
jbaena@void: ~/public_html/test
jbaena@void:~/public_html/test$ php array4.php
ciudad => Granada
pais => España
continente => Europa
Array
(
)
```

```
jbaena@void: ~/public_html/test
jbaena@void:~/public_html/test$ php array4.php
ciudad => Granada
pais => España
continente => Europa
Array
(
    [ciudad] => Granada
    [pais] => España
    [continente] => Europa
    [ciudad_x] => procesado
    [pais_x] => procesado
    [continente_x] => procesado
)
```



## Estructuras de control

### Iteradores sobre arrays

#### Iteradores sobre arrays

Cada array incluye un "puntero interno" que permite iterar sobre él

```
$nom = ["Pepe", "Ana", "Juan", "María"];
reset($nom); // Poner iterador al principio
while ($v = each ($nom)) // Obtener elemento y avanzar iterador
    echo $v['key'], ": ", $v['value'], PHP_EOL;
```

reset(\$a)	Pone iterador al principio y devuelve elemento
end(\$a)	Pone iterador al final y devuelve elemento
each(\$a)	Devuelve clave y elemento y avanza iterador
current(\$a)	Devuelve el elemento del iterador actual
key(\$a)	Devuelve la clave del iterador actual
next(\$a)	Avanza iterador y devuelve elemento
prev(\$a)	Retrocede iterador y devuelve elemento

```
reset($nom);
while ($v = next($nom))
    echo $v, PHP_EOL;
```

```
reset($nom);
while (list($k,$v) = each($nom))
    echo $k, ": ", $v, PHP_EOL;
```



UNIVERSIDAD  
DE GRANADA



## Tecnologías Web

### Grado en Ingeniería Informática

#### Programación en el lado del servidor

#### 1. El lenguaje PHP

1. Introducción
2. Variables y tipos
3. Cadenas
4. Arrays
5. Estructuras de control
- » 6. Funciones
7. Gestión de errores
8. Expresiones regulares
9. Ficheros

#### 2. PHP y aplicaciones web

#### 3. PHP y conexión con BBDD



## Funciones

### Definición, anidamiento

#### Funciones

El nombre no es sensible a mayúsculas. Por convención las funciones intrínsecas se nombran siempre en minúsculas

```
function doble($x) {
    return 2*$x;
}
```

#### Anidamiento de funciones

Es posible definir una función dentro de otra aunque:

- La función interna se puede llamar desde fuera de la función externa
- La función interna no puede llamarse hasta que se haya llamado una vez a la función externa

```
function cuarta($x) {
    function cuad($y) {
        return $y*$y;
    }
    return cuad(cuad($x));
}
echo cuarta(2), PHP_EOL;
echo cuad(2), PHP_EOL;
```



## Funciones

### Ámbito y variables

#### Ámbito de variables

Las variables definidas fuera de las funciones:

- Tienen ámbito global
- Por defecto, no son accesibles dentro de las funciones, salvo que se declaren global

Las variables definidas dentro de las funciones:

- Tienen ámbito local a la función

Las variables superglobales:

- Tienen ámbito global y local a todas las funciones
- \$GLOBALS permite acceso a globales sin declararlas global

```
$x = 4;

function f() {
    echo $x; // Error
    if (isset($_POST['usuario']))
        echo $_POST['usuario']; // Ok
}

echo $x; // Ok
f(); // Error
```

```
$x = 4;

function f() {
    global $x;
    echo $x; // Ok
}

echo $x; // Ok
f(); // Ok
```

```
$x = 4;

function f() {
    echo $GLOBALS['x']; // Ok
}

echo $x; // Ok
f(); // Ok
```



### Paso de parámetros

- Por defecto, el paso de parámetros es por valor
- El paso por referencia se hace anteponiendo & al parámetro formal

```
function doble($x) {
    $x = 2*$x;
}
$y = 2;
doble($y);
echo $y, PHP_EOL; // 2
```

```
function dobleR(&$x) {
    $x = 2*$x;
}
$y = 2;
dobleR($y);
echo $y, PHP_EOL; // 4
```

### Valores por defecto

- Se pueden indicar en la declaración de la función
- Van al final de la lista de parámetros formales

```
function dobleD($x=10) {
    return 2*$x;
}
echo dobleD(4), PHP_EOL; // 8
echo dobleD(), PHP_EOL; // 20
```



### Parámetros variables

El número de parámetros puede ser variable

```
function sumar() {
    $s=0;
    for ($i=0; $i<func_num_args(); $i++) // Número de argumentos
        $s += func_get_arg($i); // Argumento i-ésimo
    return $s;
}
```

```
echo sumar(2,5,4,6), PHP_EOL; // 17
echo sumar(1,3), PHP_EOL; // 4
echo sumar(), PHP_EOL; // 0
```

```
function sumar() {
    $s=0;
    $a=func_get_args(); // Array con todos los argumentos
    for ($i=0; $i<count($a); $i++)
        $s += $a[$i];
    return $s;
}
```



**Omitir parámetros**

Se pueden omitir parámetros en la llamada

```
function sumar3($a,$b,$c) {  
    $s=0;  
    if (isset($a))  
        $s += $a;  
    if (isset($b))  
        $s += $b;  
    if (isset($c))  
        $s += $c;  
    return $s;  
}  
  
echo sumar3(2), PHP_EOL;  
echo sumar3(2,5,7), PHP_EOL;  
echo sumar3(), PHP_EOL;
```

**Llamada previa a la definición**

Se puede llamar a una función antes de definirla

```
echo sumar3(2), PHP_EOL;  
echo sumar3(2,5,7), PHP_EOL;  
echo sumar3(), PHP_EOL;  
  
function sumar3($a,$b,$c) {  
    $s=0;  
    if (isset($a))  
        $s += $a;  
    if (isset($b))  
        $s += $b;  
    if (isset($c))  
        $s += $c;  
    return $s;  
}
```



### Funciones variables

La llamada a una función puede hacerse a través del valor de una variable

```
function suma($a,$b) {
    return $a+$b;
}
function multi($a,$b) {
    return $a*$b;
}

$oper = "suma";
echo $oper(2,4), PHP_EOL; // 6

$oper = "multi";
echo $oper(2,4), PHP_EOL; // 8
```

*// Simplifica estructuras como esta*

```
switch ($oper) {
    case "suma": echo suma(2,4), PHP_EOL;
                break;
    case "multi": echo multi(2,4), PHP_EOL;
                 break;
}
```

*// Es conveniente comprobación previa*

```
if (function_exists($oper))
    echo $oper(2,4), PHP_EOL;
```

No se puede usar para llamar construcciones del lenguaje

```
$oper = "echo";
echo $oper("Hola"); // ERROR
```



# Tecnologías Web

## Grado en Ingeniería Informática

### Programación en el lado del servidor

#### 1. El lenguaje PHP

1. Introducción
2. Variables y tipos
3. Cadenas
4. Arrays
5. Estructuras de control
6. Funciones
- » 7. Gestión de errores
8. Expresiones regulares
9. Ficheros

#### 2. PHP y aplicaciones web

#### 3. PHP y conexión con BBDD



## Gestión de errores

### Depuración

#### Depuración de errores

- Configuración de PHP
- Uso de Magic Constants

```
/* Modificación de la configuración de Apache-PHP (php.ini)*/
ini_set("display_errors","1");      /* display_errors = On */
ini_set("error_reporting",E_ALL);    /* error_reporting = E_ALL */
                                     /* E_NOTICE | E_WARNING | ... */
```

Directivas accesibles desde PHP: <http://www.php.net/manual/en/ini.list.php>

```
function mostrar_error($mensaje,$li,$fu,$fi) {
echo <<< HTML
<div class='phperror'>
<p>ERROR: $mensaje<p>
<ul>
  <li>Línea: $li</li>
  <li>Función: $fu</li>
  <li>Fichero: $fi</li>
</ul>
</div>
HTML;
}
```

```
mostrar_error("Se ha localizado el error XXXX", __LINE__, __FUNCTION__, __FILE__);
```



## Gestión de errores

### Supresión y generación de errores

#### Supresión de errores en expresiones

@ antepuesto a una expresión anula el posible mensaje de error

```
/* División por cero: provoca Warning */
echo "Dividiendo por cero ...<br>";
$x = 4 / 0;      // Muestra error
@ $x = 4 / 0;    // No muestra error

/* Acceso a elemento que no existe en array: provoca Notice */
$v['color'] = 'rojo';
echo $v['talla']; // Muestra error
echo @ $v['talla']; // No muestra error
```

#### Generar un error en tiempo de ejecución

Ejemplo de uso: comprobación de parámetros correctos en una función

```
echo "Provocando un error<br>";
trigger_error('Ha habido un error', E_USER_ERROR); // E_USER_NOTICE
                                                    // E_USER_WARNING
```

#### Finalizar ejecución de un script

```
die("He acabado");
```



### Función manejadora de errores

```
function gestionError($err, $msg, $fi, $li, $sim) {
    echo 'Código de error: ', $err, '<br>';
    echo 'Descripción del error: ', $msg, '<br>';
    echo 'Fichero del error: ', $fi, '<br>';
    echo 'Línea del error: ', $li, '<br>';
    // sim contiene tabla de símbolos (variables, etc) en el momento del error
}
```

```
set_error_handler('gestionError');
```

```
$x = 4/0;
```

Código de error: 2

Descripción del error: Division by zero

Fichero del error: /home/jbaena/Documents/Docencia/TecnologiasWeb/php/debug1.php

Línea del error: 89



### Excepciones

```
try {
    // Código que genera una excepción de tipo Error
    funcion_inexistente(3);
} catch (Error $e) { // Puede haber varios catch
    echo 'Excepción capturada: ', $e->getMessage(), '<br>';
    echo 'Código: ', $e->getCode(), '<br>';
    echo 'Fichero: ', $e->getFile(), '<br>';
    echo 'Línea: ', $e->getLine(), '<br>';
} finally { // Se ejecuta cuando acaba try o catch
    echo 'Acabado bloque try/catch<br>';
}
```

Excepciones predefinidas: <http://php.net/manual/en/reserved.exceptions.php>

Lanzar una excepción:

```
throw new Error('hay un error');
```

### 1. El lenguaje PHP

1. Introducción
2. Variables y tipos
3. Cadenas
4. Arrays
5. Estructuras de control
6. Funciones
7. Gestión de errores
8. Expresiones regulares
9. Ficheros



2. PHP y aplicaciones web
3. PHP y conexión con BBDD

#### Uso de expresiones regulares

Buscar coincidencias de patrones  
Sustituir trozos de un texto  
Dividir un texto en trozos

Las expresiones regulares están delimitadas:

- Por un carácter cualquiera
- Al principio y al final

`/^[a-z][0-9]/`

`~^[a-z][0-9]~`

`X^[a-z][0-9]X`

`(^[a-z][0-9])`

`{^[a-z][0-9]}`

El delimitador también puede ser (), [], {} o <>

Los delimitadores son escapados si forman parte de la expresión: `/http:\\\\//`  
`~http://~`

#### Buscar una coincidencia

`preg_match(patrón,cadena)`

Se busca el patrón en la cadena y devuelve true/false dependiendo de si hay emparejamiento o no

```
echo preg_match('/a/', 'Hola'), PHP_EOL; // True
echo preg_match('/x/', 'Hola'), PHP_EOL; // False
echo preg_match('/ol/', 'Hola'), PHP_EOL; // True
echo preg_match('/lo/', 'Hola'), PHP_EOL; // False
```





## Expresiones regulares

### Caracteres especiales y metacaracteres

#### Caracteres especiales

<code>\d</code>	Cualquier dígito	<code>\D</code>	Cualquier no dígito
<code>\s</code>	Espacio (blanco, tab, enter, ...)	<code>\S</code>	No espacio
<code>\w</code>	Letras, dígitos o <code>_</code>	<code>\W</code>	Ni letras, ni dígitos, ni <code>_</code>
<code>.</code>	Cualquier carácter		

```

echo preg_match('/\d\d/', '23x'), PHP_EOL; // True
echo preg_match('/\d\d/', '23xabc'), PHP_EOL; // True
echo preg_match('/\D\d/', 'x23'), PHP_EOL; // True
echo preg_match('/\D\d/', '123'), PHP_EOL; // False
echo preg_match('/x\sx/', 'x x'), PHP_EOL; // True
echo preg_match('/x\sx/', "x\tx"), PHP_EOL; // True

echo preg_match('/.../', 'Hola'), PHP_EOL; // True (3 coincidencias)
echo preg_match('/Ho.a/', 'Hola'), PHP_EOL; // True
echo preg_match('/Ho.a/', 'Hoka'), PHP_EOL; // True
echo preg_match('/...l./', 'Hola'), PHP_EOL; // True

```

#### Metacaracteres

`\ ^ $ . [ ] | ( ) ? * { }`

Tienen significado especial. Para que sean parte del patrón buscado hay que escaparlos

```

echo preg_match('/150$/','150$'), PHP_EOL; // False
echo preg_match('/150\\$/','150$'), PHP_EOL; // True

```



## Expresiones regulares

### Clases de caracteres y alternativas

#### Clases de caracteres

`[ ]` Definen conjuntos de caracteres de un tipo

```

echo preg_match('/P[aeiou]/','Hola Pepe'), PHP_EOL; // True
echo preg_match('/P[aeiou]/','Hola Lepe'), PHP_EOL; // False
echo preg_match('/c[aeiou]s/','la casa es una cosa'), PHP_EOL; // True (2)
echo preg_match('/c[aeiou]s/','esto es un caos'), PHP_EOL; // False

```

Metacaracteres dentro de `[ ]`: `^` (negación), `-` (rango)

```

echo preg_match('/c[^aeiou]p/','cup es taza'), PHP_EOL; // False
echo preg_match('/c[^aeiou]p/','c3po es un robot'), PHP_EOL; // True

echo preg_match('/c[a-k]p/','hay un cepo'), PHP_EOL; // True
echo preg_match('/c[a-k]p/','hay una copa'), PHP_EOL; // False
echo preg_match('/[4-9]/','5th'), PHP_EOL; // True
echo preg_match('/[4-9]/','2th'), PHP_EOL; // False
echo preg_match('/[a-zA-Z]/','12345'), PHP_EOL; // False

```

#### Alternativas

`|` Elegir entre 2 o más alternativas

```

echo preg_match('/Pinto|Valdemoro/','Madrid'), PHP_EOL; // False
echo preg_match('/Pinto|Valdemoro/','Pintar'), PHP_EOL; // False
echo preg_match('/Pinto|Valdemoro/','Valdemoro'), PHP_EOL; // True

```



## Expresiones regulares

### Repeticiones

#### Repeticiones

?	0 o 1 vez
*	0 o más veces
+	1 o más veces
{n}	n veces
{n,m}	Entre n y m veces (incluidos)
{n,}	Al menos n veces

```

echo preg_match('/Ma+carena/', 'Maaaaacarena'), PHP_EOL; // True
echo preg_match('/Ma+carena/', 'Mcarena'), PHP_EOL; // False
echo preg_match('/Ma?carena/', 'Mcarena'), PHP_EOL; // True
echo preg_match('/Ma?carena/', 'Maacarena'), PHP_EOL; // False
echo preg_match('/Ma*carena/', 'Mcarena'), PHP_EOL; // True
echo preg_match('/Ma*carena/', 'Maaacarena'), PHP_EOL; // True
echo preg_match('/Ma{3}carena/', 'Maaacarena'), PHP_EOL; // True
echo preg_match('/Ma{3}carena/', 'Maaaaacarena'), PHP_EOL; // False

echo preg_match('/\+[0-9]{2} [0-9]{3} [0-9]{6}/', '+34 958 123456'), PHP_EOL; // True
echo preg_match('/\+[0-9]{2} [0-9]{3} [0-9]{6}/', '+34 958123456'), PHP_EOL; // False
echo preg_match('/\+[0-9]{2} *[0-9]{3} *[0-9]{6}/', '+34 958123456'), PHP_EOL; // True
echo preg_match('/\+(\+[0-9]{2}\) [0-9]{3} [0-9]{6}/', '(+34) 958 123456'), PHP_EOL; // True
echo preg_match('/\+(\+[0-9]{2}\) +[0-9]{3} +[0-9]{6}/', '(+34) 958 123456'), PHP_EOL; // True
echo preg_match('/\+(\+[0-9]{2}\) \s*[0-9]{3} \s*[0-9]{6}/', '(+34) 958123456'), PHP_EOL; // True
echo preg_match('/\+(\+[0-9]{2}\) ?\s*[0-9]{3} \s*[0-9]{6}/', '958123456'), PHP_EOL; // False

```



## Expresiones regulares

### Agrupamientos

#### Agrupamientos o subpatrones

( ) Define un grupo dentro de una expresión

```

echo preg_match('/\(\+[0-9]{2}\)?\s*[0-9]{3}\s*[0-9]{6}/', '958123456'), PHP_EOL; // False
echo preg_match('/\(\(\+[0-9]{2}\)\)?\s*[0-9]{3}\s*[0-9]{6}/', '958123456'), PHP_EOL; // True

echo preg_match('/\(\(\+[0-9]{2}\)\)?\s*[0-9]{3}\s*[0-9]{6}/', 'x958123456x'), PHP_EOL; // True

```

#### Metacaracteres de inicio/fin de expresión

^	Debe emparejar al comienzo de la cadena
\$	Debe emparejar al final de la cadena

```

echo preg_match('/^\(\(\+[0-9]{2}\)\)?\s*[0-9]{3}\s*[0-9]{6}$/', '958123456'), PHP_EOL; // True
echo preg_match('/^\(\(\+[0-9]{2}\)\)?\s*[0-9]{3}\s*[0-9]{6}$/', '(+34) 958 123456'), PHP_EOL; // True
echo preg_match('/^\(\(\+[0-9]{2}\)\)?\s*[0-9]{3}\s*[0-9]{6}$/', 's(+34) 958 123456'), PHP_EOL; // False
echo preg_match('/^\(\(\+[0-9]{2}\)\)?\s*[0-9]{3}\s*[0-9]{6}$/', '(+34) 958 123456v'), PHP_EOL; // False

```



## Expresiones regulares

### Extracción de coincidencias

#### Obtener la(s) coincidencia(s) encontrada(s)

```
preg_match(patrón,cadena,matches)
```

```
preg_match('/c[^aeiou]p/', "c3po es un robot", $patron);
print_r($patron);      // Array ( [0] => c3p )
```

```
preg_match('/c[aeiou]s/', "la casa es una cosa", $patron);
print_r($patron);      // Array ( [0] => cas ) ... solo la primera
```

El elemento [0] es la coincidencia completa encontrada  
Cada [i] siguiente es la coincidencia de cada agrupamiento

```
preg_match('/^(\([0-9]{2}\))?\s*[0-9]{3}\s*[0-9]{6}$/',
    '(+34) 958 123456', $patron);
print_r($patron);      // Array ( [0] => (+34) 958 123456
                        //          [1] => (+34) )
```

```
preg_match('/^(\([0-9]{2}\))?\s*([0-9]{3})\s*([0-9]{6})$/',
    '(+34) 958 123456', $patron);
print_r($patron);      // Array ( [0] => (+34) 958 123456
                        //          [1] => (+34)
                        //          [2] => 958
                        //          [3] => 123456 )
```



## Expresiones regulares

### Extracción de coincidencias

#### Obtener la coincidencia encontrada en cada grupo

```
preg_match(patrón,cadena,matches)
```

?: permite omitir algunos grupos

```
preg_match('/^(?:\([0-9]{2}\))?\s*[0-9]{3}\s*(?:[0-9]{6})$/',
    '(+34) 958 123456', $patron);
print_r($patron);      // Array ( [0] => (+34) 958 123456
                        //          [1] => 958 )
```

#### Referencias a patrones previos

```
\1 \2 \3 ...
```

```
preg_match('/^(\([0-9]{2}\))?\s*[0-9]{3}\s*(\2\2)$/',
    '(+34) 958 123456', $patron);
print_r($patron);      // No encuentra
```

```
preg_match('/^(\([0-9]{2}\))?\s*[0-9]{3}\s*(\2\2)$/',
    '(+34) 958 958958', $patron);
print_r($patron);      // Array ( [0] => (+34) 958 958958
                        //          [1] => (+34)
                        //          [2] => 958
                        //          [3] => 958958 )
```



## Expresiones regulares

### Extracción de coincidencias y sustitución

#### Obtener todas las coincidencias encontradas

```
preg_match_all(patrón,cadena,matches)
```

```
preg_match('/c[aeiou]s/', "la casa es una cosa", $patron);
print_r($patron); // cas (la primera)
```

```
preg_match_all('/c[aeiou]s/', "la casa es una cosa", $patron);
print_r($patron); // cas cos (todas)
```

#### Sustituir coincidencias encontradas

```
preg_replace(patrón,sustitución,cadena)
```

```
$result = preg_replace('/\s+/', '-', '(+34) 958 123456'); // (+34)-958-123456
```

```
$nombres = ["Pepito Pérez", "José García", "Ana Martín"];
$nuevos = preg_replace('/(\w)\w* (\w+)/', '\1. \2', $nombres);
print_r($nuevos); // P. Pérez, José García, A. Martín
```

```
$nuevos = preg_replace('/(\w)\w* (\w+)/u', '\1. \2', $nombres);
print_r($nuevos); // P. Pérez, J. García, A. Martín
```

\w = caracteres para formar palabras [0-9a-zA-Z]  
u: considerar caracteres unicode



## Expresiones regulares

### Separación de cadenas

#### Separación de cadena por patrones

```
preg_split(patrón,cadena)
```

```
$trozos = preg_split('/\s/', 'Antonio Pérez García');
print_r($trozos); // "Antonio", "Pérez", "García"
```

#### Para obtener la posición de cada trozo separado

```
$trozos = preg_split('/[\s,]+/', 'Pérez García, Antonio',
-1, PREG_SPLIT_OFFSET_CAPTURE);
print_r($trozos); // [{"Pérez",0}, {"García",7}, {"Antonio",16}]
```

#### Para obtener también los separadores

```
$trozos = preg_split('/([\s,]+)/', 'Pérez García, Antonio',
-1, PREG_SPLIT_DELIM_CAPTURE);
print_r($trozos); // ["Pérez", " ", "García", " ", " ", "Antonio"]
```

```
$trozos = preg_split('#([+/*])#', '4+2*7/5', -1, PREG_SPLIT_DELIM_CAPTURE);
print_r($trozos);
```

#### 1. El lenguaje PHP

1. Introducción
2. Variables y tipos
3. Cadenas
4. Arrays
5. Estructuras de control
6. Funciones
7. Gestión de errores
8. Expresiones regulares
9. Ficheros



2. PHP y aplicaciones web
3. PHP y conexión con BBDD

#### Apertura de un fichero

```
$f=fopen($nombre,$modo)
```

Si hay éxito devuelve un recurso asociado al fichero y si no FALSE

\$nombre: nombre del fichero, puede ser una URL

#### Modos de apertura habituales

r	Lectura (puntero al comienzo)
r+	Lectura/escritura (ptr al comienzo)
w	Escritura. Crea o trunca fichero si existe (ptr al comienzo)
w+	Lectura/escritura. Crea o trunca fichero si existe (ptr al comienzo)
a	Escritura (ptr al final). Si no existe se crea
a+	Lectura/Escritura (ptr al final). Si no existe se crea

#### Cierre de un fichero

```
fclose($f)
```

#### Comprobaciones

file_exists(\$nombre)	Comprueba existencia
is_readable(\$nombre)	Comprueba si existe y es legible
is_writable(\$nombre)	Comprueba si existe y es escribible
feof(\$f)	Comprueba si ptr está al final del archivo




**Lectura de datos**

<code>fgets(\$f)</code>	Lee una línea completa
<code>fgetc(\$f)</code>	Lee un carácter
<code>fread(\$f,\$tam)</code>	Lee un número de bytes

```
<?php
```

```
// Abrir fichero
```

```
$f = fopen("datos.txt","r") or die("Error en apertura");
```

```
while (!feof($f)) {
    $cad = fgets($f);
    echo "L (", strlen($cad), "): ", $cad;
}
```

```
fclose($f);
```

```
?>
```

datos.txt

Línea 1  
Línea 2  
Línea 3



L (8): Línea 1  
L (9): Línea 2  
L (8): Línea 3


**Lectura de datos**

`$cad = file_get_contents($fnombre)`  
 Carga el contenido el fichero y lo devuelve como una cadena  
 El nombre puede ser una URL  
 Si falla la lectura devuelve FALSE

```
$cad = file_get_contents("datos.txt") or die("Error en apertura");
echo "Tamaño: ", strlen($cad), PHP_EOL;
echo "Cadena: ", $cad, PHP_EOL;
```

`$vec = file($fnombre)`  
 Carga el contenido el fichero y lo devuelve como un vector de cadenas. Cada elemento del vector es una línea del fichero.  
 Si falla la lectura devuelve FALSE

```
$cads = file("datos.txt") or die("Error en apertura");
foreach ($cads as $c)
    echo "L (", strlen($c), "): ", $c;
```



### Lectura de datos

```
readfile($nombre)
```

Lee el contenido el fichero y lo envía a la salida  
 El nombre puede ser una URL  
 Devuelve el número de bytes leídos o FALSE

hola.html

```
<!DOCTYPE html>
<html> <head>
<meta charset="utf-8">
<title>Tecnologías Web</title>
</head> <body>
<h1>Hola mundo</h1>
</body> </html>
```

```
readfile("hola.html") or die("Error en apertura");

// ... o bien ...
$cad = file_get_contents("hola.html") or die("Error en apertura");
echo $cad;
```



### Escritura de datos

```
fwrite($f,$cad)
```

Escribe el contenido de \$cad en el fichero

```
file_put_contents($nombre,$cad)
```

Escribe el contenido de \$cad en el fichero. Por defecto sobrescribe

```
file_put_contents($nombre,$cad,FILE_APPEND)
```

En este caso añade contenido al fichero

```
$f = fopen("salida.txt", "w") or die("Error en apertura");
fwrite($f,"Línea 1\n");
fwrite($f,"Línea 2\n");
fwrite($f,"Línea 3");
fclose($f);

file_put_contents("salida.txt", "Línea 1\nLínea 2\nLínea 3");
```

file\_put\_contents equivale a fopen + fwrite + fclose



### Bloqueo de acceso concurrente

`flock($f,$modo)` Establece el modo de bloqueo de un fichero

#### Modos de bloqueo:

<code>LOCK_SH</code>	Bloqueo compartido (lectura)
<code>LOCK_EX</code>	Bloqueo exclusivo (escritura)
<code>LOCK_UN</code>	Desbloqueo
<code>LOCK_NB</code>	Modo no-bloqueante. Se combina con los anteriores

```
$f = fopen("datos.txt", "r+");
if (flock($f, LOCK_EX)) { // adquirir un bloqueo exclusivo
    // Podemos leer/escribir
    // ...
    flock($f, LOCK_UN); // libera el bloqueo
} else
    echo "No se puede acceder al recurso";
fclose($f);
```

Si un fichero está bloqueado: `flock` espera hasta que se desbloquee

```
if (flock($f, LOCK_EX | LOCK_NB)) {
    // adquirir un bloqueo exclusivo, si no se puede continua la ejecución
}
```



### Más operaciones sobre ficheros

<code>copy</code>	Copia un fichero en otro
<code>rename</code>	Cambia el nombre de un fichero
<code>unlink</code>	Borra un fichero
<code>is_file</code>	Comprueba si es un fichero
<code>is_dir</code>	Comprueba si es un directorio
<code>mkdir</code>	Crea un directorio
<code>filesize</code>	Devuelve tamaño de un fichero
<code>fseek</code>	Posiciona el puntero del fichero
<code>ftell</code>	Devuelve la posición del puntero del fichero
<code>tmpfile</code>	Crea un archivo temporal y lo abre
<code>tempnam</code>	Crea un archivo temporal en una carpeta especificada



### CSV: Comma separated values

Útiles para proceso de datos por lotes, backups, etc.

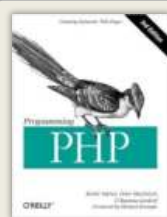
`fgetcsv` Lee una línea y devuelve los campos en un array de cadenas

`fputcsv` Almacena un array en una línea con formato CSV

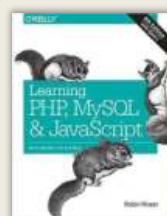
`str_getcsv` Convierte un string a un array de strings

```
<?php
$f=fopen("datos.csv","r") or die("Error en apertura");
if ( ($cab=fgetcsv($f,0,',')) != FALSE) {
    $dat = [];
    while ( ($d=fgetcsv($f,0,',')) != FALSE) {
        // Añadimos a array
        $dat[] = $d;
        // ... o cualquier otro tipo de procesamiento
    }
}
print_r($cab);
print_r($dat);
fclose($f);
?>
```

Nombre;Apellidos;DNI  
 Pepe;Pérez Martín;12312312Q  
 Ana;González Pérez;21212121R  
 Juan María;Gómez Moral;32132132W  
 José;Aranda Garrido;54345434T

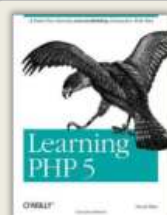


Kevin Tatroe, Peter MacIntyre, Rasmus Lerdorf  
**Programming PHP**  
 O'Reilly. 2013



Robin Nixon  
**Learning PHP, MySQL, & JavaScript (4th ed)**  
 O'Reilly. 2014

<http://lpmj.net/>



David Sklar  
**Learning PHP.**  
**A gentle introduction to the web's most popular language**  
 O'Reilly. 2016